

Master Thesis

Max Neuwirt

An Intent-Based Architecture for the Dynamic
Construction of Multimodal Large Language Models

Max Neuwirt

An Intent-Based Architecture for the Dynamic Construction of Multimodal Large Language Models

Master thesis submitted for examination in Master's degree
in the study course *Master of Science Informatik*
at the Faculty of Computer Science and Digital Society
at University of Applied Science Hamburg

Supervisor: Prof. Dr. Kai von Luck

Supervisor: Prof. Dr. Peer Stelldinger

Submitted on: 24. August 2026

Max Neuwirt

Thema der Arbeit

Eine intent-basierte Architektur zur dynamischen Konstruktion multimodaler Large Language Models

Stichwörter

Multimodale Large Language Models, Intent-basierte Systeme, deklarative Modellkomposition, parametereffizientes Training

Kurzzusammenfassung

Multimodale Large Language Models (MLLMs) erweitern die Fähigkeiten großer Sprachmodelle um das Verständnis nicht-textueller Eingaben wie Bilder und Audio. Obwohl das Encoder-Bridge-LLM-Muster theoretisch modular ist, legen veröffentlichte Modelle, mit dieser Architektur, die Wahl von Encoder, Bridge und Sprachmodell im Code fest, sodass jeder Komponententausch und jede Modalitätserweiterung manuelle Implementierungsarbeit erfordert. Diese Arbeit überträgt das Paradigma intent-basierter Systeme auf die Konstruktion von MLLMs. Entworfen und implementiert wird eine Plattform aus zwei entkoppelten Komponenten. Der MLLM Designer, ein visueller Node-Graph-Editor, erstellt eine Architektur aus Encoder-, Projektions-, Datensatz- und LLM-Knoten und exportiert diese als deklarative JSON-Konfiguration. Die Engine liest diese Konfiguration und setzt das Modell aus vortrainierten Komponenten zusammen. Dabei werden ausschließlich die Projektionsschichten bei eingefrorenen Encodern und LLM trainiert. Validiert wird die Plattform durch drei Experimente: Einen kontrollierten Encoder-Tausch von CLIP zu SigLIP-2, die Erweiterung eines reinen Vision-Modells um eine Audio-Modalität und eine Reproduktion der LLaVA-1.5-Stage-1-Architektur. Alle drei Architekturänderungen ließen sich allein durch Konfigurationsänderungen durchführen. Die reproduzierte LLaVA-1.5-Stage-1-Architektur lag auf ScienceQA und MM-Bench bei jedem Seed über dem Zufall. Der Encoder-Tausch verbesserte den MMBench-Score, während die Audio-Erweiterung die Beschreibungsqualität bei gemeinsamer Inferenz verschlechterte. Deklarative Komposition liefert damit funktionsfähige Modelle, jedoch bedeutet dies nicht, dass sie immer die beste Leistung erbringen. Abschließend kann gesagt werden, dass die Plattform den Aufwand der MLLM-Konstruktion von der Implementierung zur experimentellen Validierung des Entwurfs verlagert.

Max Neuwirt

Title of Thesis

An Intent-Based Architecture for the Dynamic Construction of Multimodal Large Language Models

Keywords

Multimodal Large Language Models, Intent-Based Systems, declarative model composition, parameter-efficient training

Abstract

Multimodal Large Language Models (MLLMs) extend the reasoning abilities of Large Language Models (LLMs) to non-textual inputs such as images and audio. Although the prevailing encoder-bridge-LLM pattern is conceptually modular, published models fix the choice of encoder, bridge, and language model in code, so every component swap or modality extension requires manual implementation work. This work transfers the paradigm of intent-based systems to the construction of MLLMs. A platform of two decoupled components is designed and implemented. The MLLM Designer, a visual node-graph editor, composes an architecture from encoder, projection, dataset, and LLM nodes and exports it as a declarative JSON configuration. The Engine ingests this configuration, assembles the model from pre-trained components, trains only the projection bridges while keeping encoders and LLM frozen. The platform is validated through three experiments, a controlled encoder replacement from CLIP to SigLIP-2, the extension of a vision-only model with an audio modality, and a reproduction of the LLaVA-1.5 Stage-1 architecture. All three architectural changes were expressed purely through configuration edits. The reproduction of the LLaVA-1.5 Stage-1 architecture scored above chance on ScienceQA and MMBench under every seed. The encoder swap improved the MMBench score, while the audio extension degraded caption quality under joint inference. Declarative composition therefore reliably yields functional models, yet this does not mean that those models always achieve the best performance. Finally, the platform shifts the effort of MLLM construction from implementation towards experimental validation of the composed design.

Contents

List of Figures	viii
List of Tables	ix
Listings	xi
1 Introduction	1
1.1 Problem Statement	2
1.2 Research Questions	2
1.3 Approach and Contributions	3
1.4 Thesis Outline	3
2 Foundations	4
2.1 Large Language Models	4
2.1.1 Transformer	4
2.1.2 Embedding Space	5
2.1.3 Tokenization	6
2.1.4 Likelihood-Based Scoring	6
2.2 Multimodal Large Language Models	8
2.3 Intent-Based Systems	9
2.4 Declarative Deployment Model	10
3 Related Work	12
3.1 Parameter-Efficient Fine-Tuning for Modality Adaptation	12
3.2 Modular Architectures with Bridging Mechanisms	16
3.3 Unified Embedding Spaces	19
3.4 LLM-as-Controller Frameworks	20
3.5 Summary	21

4	System Design	23
4.1	Architectural Paradigm: Intent-Based Architecture	23
4.2	The MLLM Designer	24
4.3	The Engine	29
5	Experiments	32
5.1	Experimental Setup	32
5.1.1	Training Within Hardware Constraints	36
5.1.2	Scoring Protocols	37
5.2	Experiment 1: Component Replacement	38
5.2.1	Motivation and Objective	38
5.2.2	Hypotheses	38
5.2.3	Procedure	39
5.2.4	Benchmarks	39
5.2.5	Results	40
5.3	Experiment 2: Multi-Modality Expansion	43
5.3.1	Motivation and Objective	43
5.3.2	Hypotheses	43
5.3.3	Procedure	43
5.3.4	Benchmarks	44
5.3.5	Results	45
5.4	Experiment 3: Reproducing a Published Architecture	48
5.4.1	Motivation and Objective	48
5.4.2	Hypotheses	48
5.4.3	Choice of Model	48
5.4.4	Procedure	50
5.4.5	Benchmarks	50
5.4.6	Results	51
6	Discussion	55
7	Limitations and Outlook	61
7.1	Limitations	61
7.2	Outlook	63
	Bibliography	66

A Appendix	76
A.1 Tools Used	76
A.2 Full JSON Configuration Example	76
Declaration of Authorship	79

List of Figures

2.1	The modular MLLM architecture with multiple encoders, projection layers, and a shared LLM. Each encoder produces feature embeddings that a dedicated projection layer aligns to the LLM’s embedding space before a concatenation module merges all aligned representations into a single input sequence.	8
3.1	Architectural comparison between the encoder-based MLLM approach and VoRA [68]. The encoder-based architecture on the left routes image patches through a ViT and a connector before the LLM, whereas VoRA on the right replaces both components with LoRA layers inside the LLM.	15
4.1	Overview of the system architecture with its two components, the MLLM Designer and the Engine. The Designer serializes the visual graph into a JSON intent and sends it to the Engine through a REST API. The Engine validates the intent, assembles the model, trains it, and exports the result as a runtime bundle or Hugging Face model directory.	23
4.2	Wireframe of the MLLM Designer interface. The canvas shows encoder, projection, dataset, and LLM nodes connected as a directed graph, with a node palette at the bottom and build controls on the right.	24
4.3	The implemented MLLM Designer with a two-modality pipeline on the canvas. A vision encoder and an audio encoder each connect through a dedicated projection node to a shared LLM, with dataset nodes supplying the training data for each projection layer.	26

List of Tables

5.1	Training configuration used across all experiments. The step cap applies to Experiments 1 and 2 only.	33
5.2	Model and dataset identifiers, pinned to Hugging Face commits, used across all experiments.	35
5.3	Likelihood-based MMBench accuracy for Experiment 1 under seeds 2, 42, and 534. The baseline configuration uses <code>openai/clip-vit-large-patch14-336</code> and the swap configuration <code>google/siglip2-so400m-patch14-384</code> , with all user-specified settings held identical and the projection layer adapting automatically to the encoder’s output dimension. The measured chance level of the single-pass <code>dev</code> evaluation is 0.266.	41
5.4	Mean MMBench accuracy over seeds 2, 42, and 534 under four scoring conditions. Trained is the standard evaluation with the learned projection. Shuffled feeds another question’s image. Text-only suppresses the visual prefix entirely. Random projection passes the real image through a re-initialized projection. Chance is <code>mean(1/n_options)</code> over the evaluated questions. The text-only result is identical across all six bundles because suppressing the prefix collapses each bundle to its bare LLM.	42
5.5	BLEU-1 scores on the <code>mteb/MSR-VTT</code> test split for Experiment 2 under seeds 2, 42, and 534. The two lower rows evaluate the identical trained Vision + Audio model, once restricted to vision inputs and once with the audio track additionally provided, so their per-seed difference isolates the contribution of the audio input at inference time.	46

5.6	Likelihood-based benchmark accuracies for Experiment 3, averaged over seeds 42, 2, and 534. The chance baselines are measured as the mean number of answer options over the evaluated questions (see Section 5.1.2). The published LLaVA-1.5 (7B) scores are included for orientation only, as they were obtained after the full two-stage training pipeline and under a generation-based protocol.	52
5.7	Mean likelihood-based accuracy over seeds 2, 42, and 534 under four scoring conditions. The text-only result is identical across all three seeds because suppressing the visual prefix collapses each to the bare Vicuna-7B LLM.	53
A.1	Tools and Aids Used	76

Listings

4.1	Simplified JSON configuration exported from the MLLM Designer, showing a vision path.	28
A.1	Complete JSON configuration exported from the two-modality design shown in Figure 4.3.	76

1 Introduction

Large Language Models (LLMs) have established themselves as general-purpose task solvers, exhibiting capabilities such as in-context learning, instruction following, and multi-step reasoning over text [85, 52]. Their success has motivated the development of Multimodal Large Language Models (MLLMs), which extend the reasoning abilities of an LLM to non-textual inputs such as images and audio [78, 69]. The prevailing architectural pattern for constructing such systems is modular [67]. A pre-trained modality encoder extracts features from the non-textual input, a lightweight projection layer maps these features into the embedding space of the language model, and the LLM performs reasoning and response generation over the combined input [78, 81]. Because the encoder and the LLM remain frozen while only the projection layer is trained, this pattern promises the free interchangeability of its components [81, 34].

In practice, this interchangeability is not operational. Published MLLMs are delivered as self-contained code bases in which the choice of encoder, projection layer, and LLM is fixed in model code and training scripts. Replacing a component or adding a modality therefore means modifying that implementation, and as the review in Section 3 shows, nearly all current integration methods require explicit code changes for such modifications. Beyond the model code itself, considerable engineering effort is embedded in the surrounding integration logic. Embedding dimensions must be matched between encoder and LLM, each encoder expects its own preprocessor and input format, datasets follow no common schema, and the training loop must be adapted to every architectural variation. Composing or modifying an MLLM consequently demands both deep learning expertise and substantial implementation work, even when the intended change is conceptually as simple as exchanging one encoder for a newer one.

A structurally similar problem has been addressed in systems and network operation through Intent-Based Systems (IBS) [19]. There, the operator declares “what” the system should look like in a declarative specification, and the system itself translates this

intent into the imperative steps that realize it, a paradigm popularized at scale by Kubernetes [9]. This thesis transfers this paradigm to the construction of MLLMs. If the architecture of an MLLM can be captured as an intent, then composing, extending, and modifying such models reduces to editing a configuration rather than a code base.

1.1 Problem Statement

This thesis addresses the lack of an accessible framework for composing MLLMs from interchangeable components without writing training code. The modular encoder–projection–LLM pattern is conceptually plug-and-play, yet no available platform allows a user to declare such an architecture, hand the declaration to a system, and receive a trained, executable model in return. Every component swap, modality extension, or architectural variation instead requires hand-written code changes to the model and training code, which restricts systematic architectural experimentation to those who can afford this implementation effort.

1.2 Research Questions

This work is guided by two research questions. The primary question, which constitutes the central contribution of this thesis, is:

RQ1: Can a modular, intent-based platform be designed and implemented that enables the composition, extension, and component-by-component replacement of MLLMs without direct code changes?

A secondary question asks whether the models produced by such a platform are functionally viable, thereby serving as a validation of the platform rather than as a claim of state-of-the-art model performance:

RQ2: Do the models produced by the platform yield functionally adequate outputs, in the sense of scoring measurably above chance on multimodal evaluation tasks?

The aim of this work is to design, prototype, and validate such a platform.

1.3 Approach and Contributions

To answer these research questions, a platform consisting of two components was designed and implemented as part of this thesis.

The contributions of this thesis are the following:

- **An intent-based architecture for MLLM construction.** The Intent Fulfillment phase of IBS is mapped onto MLLM composition, with a declarative JSON configuration acting as the single source of truth from which models are built (see Section 4).
- **The MLLM Designer.** A React Flow-based visual editor through which MLLM architectures are composed via drag-and-drop and serialized into the intent configuration without exposing any model code to the user.
- **The Engine.** A backend that translates the intent into an executable model, trains the projection layers on specified data, and exports the result as a runtime bundle that runs independently of the platform.
- **An experimental validation of declarative MLLM composition.** Three experiments demonstrate component replacement, modality extension, and the reproduction of a published architecture entirely through configuration edits (see Section 5).

1.4 Thesis Outline

The remainder of this thesis is structured as follows. Section 2 introduces the technical foundations, covering LLMs and the Transformer architecture, embedding spaces, MLLMs, and the intent-based paradigm together with its declarative deployment model. Section 3 reviews related work on integrating non-textual modalities into LLMs and categorizes existing approaches into parameter-efficient fine-tuning, modular bridging architectures, unified embedding spaces, and LLM-as-controller frameworks. Section 4 presents the architecture of the platform, detailing the intent-based design, the MLLM Designer, and the Engine. Section 5 describes the three experiments conducted to validate the platform, including their setup and results. Section 6 interprets these results with respect to the research questions. Section 7 concludes this work by describing the limitations of the prototype and outlining directions for future work.

2 Foundations

2.1 Large Language Models

LLMs represent a major advance in artificial intelligence (AI), defined as transformer-based models containing billions of parameters, pre-trained on massive text datasets [52, 85]. Functioning as statistical language models, LLMs encode a probability distribution over sequences of words, allowing them to predict linguistic patterns [7]. The underlying Transformer architecture (see Section 2.1.1) is key to their success, as it relies on attention mechanisms instead of recurrence, enabling efficient handling of large token sequences and strong capabilities in language understanding, generation, and problem-solving [10, 8]. Models like the GPT [53] and LLaMA [64] series exhibit emergent abilities, such as in-context learning, which allows the model to perform new tasks by learning from examples presented directly in the prompt without parameter updates [52, 14], instruction following, and multi-step reasoning, also called Chain-of-Thought, which are absent in smaller models and establish them as general-purpose task solvers [85]. These capabilities allow LLMs to be used across diverse applications, ranging from automated translation and summarization to domain-specific tasks like generating code or acting as the core reasoning engine within MLLMs (see Section 2.2).

2.1.1 Transformer

The 2017 paper “Attention Is All You Need” introduced the Transformer architecture, which replaced recurrence-based sequence modeling [66]. Prior to this, sequence-to-sequence tasks like machine translation or language understanding were dominated by Recurrent Neural Networks [24], Long Short-Term Memory networks [31] and Convolutional Neural Networks [39]. These architectures faced limitations regarding sequential computation and long-range dependencies.

The Transformer removed recurrence and convolution entirely, relying instead on a self-attention mechanism [66]. Its architecture consists of an encoder-decoder stack that uses layered, repeated blocks containing a Multi-Head Self-Attention mechanism and a position-wise Feed-Forward Network [66]. The central innovation is “Scaled Dot-Product Attention”, which allows the model to attend to the entire sequence at once, determining the influence each element has on others without regard to its distance in the sequence [66]. This architecture has been shown to surpass state-of-the-art performance while being more parallelizable and requiring less time to train [66], becoming the foundation for modern LLMs.

2.1.2 Embedding Space

An embedding in machine learning is a numerical representation of discrete, high-dimensional data, like words or images, as continuous, lower-dimensional vectors [51, 50]. Embeddings convert non-numerical, symbolic data into a numerical format that neural networks can process, since machine learning models operate exclusively on numerical data [59]. An embedding space or latent space is an embedding of a set of data items, for example words [51, 50].

The fundamental principle of an embedding space is that semantic relationships are encoded as spatial relationships. Items with similar meanings or contexts are mapped to vectors that are close to each other in this new vector space (e.g., as measured by cosine similarity or Euclidean distance) [51].

For example, in a well-trained word embedding space, the vector for “king” would be located near the vector for “queen,” and the vector for “walking” would be near “running” [51, 12].

Embedding spaces are a component in nearly all modern AI systems, including the Transformer architecture.

1. **Enabling Numerical Computation:** Models like the Transformer cannot operate on raw text. An embedding layer is the first step in these models, converting input tokens (words) into their corresponding vector representations [12].
2. **Efficiency and Generalization:** By representing concepts in a dense vector space, models can generalize. They can understand that “king” and “queen” are

related without having to see every possible context for both. This semantic similarity is what allows models to understand and generate coherent text.

For MLLMs, the core technical challenge involves transforming feature vectors extracted from diverse modalities into this unified feature space that the LLM understands [69, 78]. This cross-modal conversion is achieved via specialized connectors or bridges, such as single linear projection layers, Multi-Layer Perceptrons (MLPs), or more complex modules like the Q-Former, which align the non-textual features with the LLM’s embedding space [81, 8]. The effectiveness of the MLLM depends on integrating these diverse representations within a shared latent space so that the LLM can reason over them and generate responses [10].

2.1.3 Tokenization

Tokenization converts raw text into sequences of discrete units, known as tokens, that an LLM can process [7, 85]. Tokens serve as meaningful components of words, replacing the simple word-based segmentation traditionally employed in Natural Language Processing [23]. Modern LLMs heavily rely on subword tokenizers, such as Byte-Pair Encoding (BPE) [60], to mitigate the challenge of out-of-vocabulary words and manage vocabulary size effectively [85, 52]. For instance, the LLaMA models utilize a BPE algorithm implemented through SentencePiece [38], splitting numbers into individual digits and falling back to bytes for decomposing unknown UTF-8 characters [64]. Once input text is tokenized and converted into vector representations (embeddings), the Transformer architecture necessitates the injection of absolute or relative positional encodings to inform the model about the sequence order, as the design itself lacks recurrence or convolution [66]. For MLLMs, the same idea is extended to non-textual modalities. Feature vectors extracted from images or audio are transformed into a sequence of tokens and projected into the LLM’s shared embedding space, using projection layers as a compatible visual tokenizer [44, 78].

2.1.4 Likelihood-Based Scoring

Because an autoregressive LLM models a probability distribution over token sequences, it can be used not only to generate text but also to score a given text by measuring how probable the model considers it. This scoring capability is the basis for

likelihood-based evaluation, which was introduced for evaluating base language models on multiple-choice tasks with GPT-3 [14] and formalized in detail by Biderman et al. [13].

Given a prompt $x = x_0, x_1, \dots, x_{n-1}$ of n tokens and a candidate continuation $y = y_0, y_1, \dots, y_{m-1}$ of m tokens, the conditional log-probability $\log P(y \mid x)$ is computed in a single forward pass. The prompt and continuation are concatenated into a single sequence, with the final token y_{m-1} discarded, yielding $x_0, x_1, \dots, x_{n-1}, y_0, y_1, \dots, y_{m-2}$. This sequence is passed through the model, which returns a logit vector at every position. Because the model predicts the next token at each position, the logit vector at position $n - 1$ (the last prompt token) already contains the model’s prediction for y_0 , the logit vector at position n contains the prediction for y_1 , and so on up to the prediction for y_{m-1} at position $n + m - 2$. The final token y_{m-1} is therefore not included in the input, as its only effect would be to produce a prediction for a token beyond the continuation, which is not needed. A log-softmax is applied to the logit vectors at these m positions to obtain log-probabilities, and the conditional log-likelihood of the full continuation is the sum over the individual token log-probabilities [13]:

$$\log P(y \mid x) = \sum_{i=0}^{m-1} \log p(y_i \mid x, y_0, \dots, y_{i-1}) \quad (2.1)$$

As a concrete example, consider a model that is prompted with “The capital of France is” and must score the candidate continuation “Paris”. If the tokenizer splits “Paris” into two tokens, `Par` and `is`, then $m = 2$. The model receives the concatenated input “The capital of France is `Par`” (the last continuation token `is` is discarded), and in a single forward pass produces a prediction at every position. The log-probability of `Par` given the prompt and the log-probability of `is` given the prompt and the preceding token `Par` are read from the corresponding positions and summed to yield $\log P(\text{“Paris”} \mid \text{“The capital of France is”})$.

For a multiple-choice task with k answer options $a_1, \dots, a_k$, this procedure is repeated for each option, and the model’s prediction is the option with the highest conditional log-probability [14]. When answer options differ in token length, the raw log-likelihoods can be normalized by dividing each score by the number of tokens in the respective option, so that longer answers are not penalized solely for containing more terms [14, 13].

2.2 Multimodal Large Language Models

MLLMs extend LLMs by integrating diverse data types such as text, images, audio, and video within a single system, a principle referred to as multimodality [8, 69]. While LLMs possess reasoning and language generation abilities, they are inherently limited to textual input, being unable to process and understand visual and other sensory information [7, 17]. The goal of developing an MLLM is to overcome this limitation, allowing the model to perceive and reason across modalities for applications like visual question answering (VQA) and multimodal dialogue [7, 43]. There are multiple approaches to enable an LLM to understand multimodal input as discussed in Section 3. A common architectural approach to bridging the gap between a non-textual modality and the LLM involves introducing an alignment mechanism tasked with transforming modality-specific features into a unified embedding space (see Section 2.1.2) comprehensible to the language model [78]. This approach makes use of three components: a pre-trained modality encoder, a bridge, and an LLM [78], as visualized in Figure 2.1.

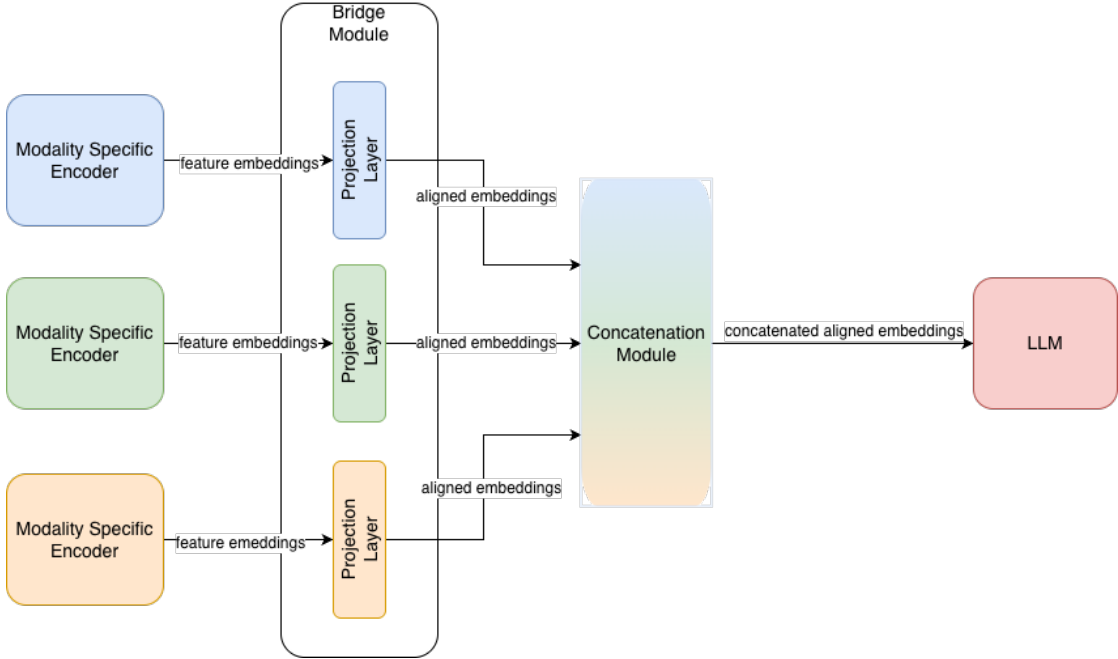


Figure 2.1: The modular MLLM architecture with multiple encoders, projection layers, and a shared LLM. Each encoder produces feature embeddings that a dedicated projection layer aligns to the LLM’s embedding space before a concatenation module merges all aligned representations into a single input sequence.

The encoder operates in a frozen state to preserve its domain-specific knowledge [7, 34]. Implementations for the bridge range from simple trainable linear projection layers or MLPs, which map feature dimensions, to more complex query-based modules like the Q-Former [42], which use attention to abstract and compress information into a fixed number of tokens [81, 63]. The aligned multimodal tokens are concatenated with text embeddings (see Section 2.1.2) and given to the LLM for reasoning and autoregressive response generation. This modular approach makes it possible to add new modalities to an MLLM and to update its core components, while keeping training cost-effective by focusing optimization on the lightweight bridge components [81, 34].

2.3 Intent-Based Systems

IBS aim to automate system design and operation by letting users specify outcomes rather than implementation steps [9]. One popular field of operation for IBS is Intent-Based Networking (IBN), which applies these concepts to network management to define system behavior based on high-level business objectives rather than complex technical configurations [28].

Intent is defined as a set of operational goals a network is supposed to meet and outcomes it is supposed to deliver [11]. Crucially, intent is defined in a declarative manner, meaning it specifies “what” to accomplish, without specifying “how” to achieve or implement it [19].

By focusing on the desired outcome rather than the steps to reach it, intent enables greater operational efficiency and flexibility, reducing dependency on human activity and the possibility of mistakes [19, 80]. An IBS is therefore a system that recognizes and ingests this high-level intent, configures itself accordingly, and determines the necessary steps to realize the given goal.

IBS are defined by a Declarative Deployment Model (DDM, see Section 2.4), an architectural pattern that focuses on achieving high-level goals or intent, expressed in a declarative manner that avoids specifying the detailed implementation steps [19]. The system is fundamentally structured as a closed-loop automation system that continuously seeks to satisfy and maintain this declarative state [79, 40]. This framework is partitioned into two phases: Intent Fulfillment and Intent Assurance [19]. The Intent Fulfillment phase is responsible for taking the intent from its origin, for example a user,

to its realization in a system or network. This phase begins with the User Space or Intent Interface, which is the first set of functions that interface with a human and are concerned with Intent Ingestion and Interaction with Users [19, 28]. This interface serves as the input layer and bridge between business objectives and automated operations, allowing users to articulate and set needed goals [28]. Intent Fulfillment then ends with a Translation Engine to transform the high-level requirements into concrete, executable configurations and executing them to establish the desired state in the system or underlying infrastructure [28, 25]. Following the Intent Fulfillment phase, the Intent Assurance phase ensures compliance by constantly monitoring the actual system behavior against the specified intent, which serves as the system’s Single Source of Truth (SSoT) [19]. The Assurance functions report status back to the User Space, thereby completing the intent control loop [19]. If the operational state deviates from the intent, the system initiates actions to restore alignment with the intent, enabling self-management and optimization throughout the system’s life cycle [79, 19].

2.4 Declarative Deployment Model

The theoretical foundation for achieving intent fulfillment lies in the distinction between declarative and imperative modeling approaches.

The DDM is the architectural pattern used within IBS to formalize user intent into an executable state representation [28]. The DDM focuses exclusively on specifying “what” the end result or the desired state should be [25]. DDM uses structural models that are interpreted by a deployment and management system to enforce this desired state [25].

In contrast, the imperative model specifies “how” to accomplish a task by defining a sequence of steps, their execution order, and the data flow [25]. While imperative modeling offers high customization, it demands significantly more expertise [25].

Because intent is a declarative specification of the desired outcome, Intent-Based Architecture is built upon and functions as a subset of the declarative model paradigm. A key architectural benefit of DDM is its inherent idempotency. This approach allows the DDM to function as the SSoT for the configuration of a system.

A widely recognized industry example of the declarative model is Kubernetes [56, 9]. Kubernetes’s standardization and automation tools, which are used for the deployment,

maintenance, and convergence of systems to desired states, have been important in enabling modern automation.

3 Related Work

The integration of multiple modalities, particularly vision and language, into LLMs has become a popular research field in AI [78, 69, 34]. The primary challenge lies in enabling text-centric LLMs to understand and reason about fundamentally different data types such as pixels and audio waveforms [78, 81].

To address this, recent research has explored a spectrum of architectural and methodological strategies. These strategies range from parameter-efficient fine-tuning (PEFT) techniques, which insert small, trainable modules into a frozen LLM, to modular frameworks using bridges to connect external encoders, to approaches where the LLM acts as a controller orchestrating external specialized models. This chapter reviews the key architectural and methodological trends, categorizing them by their core integration approach.

3.1 Parameter-Efficient Fine-Tuning for Modality Adaptation

PEFT is originally a method to train an LLM for a specific downstream task or to enhance its text-based knowledge in a given domain. With PEFT, the majority of the LLM’s parameters are frozen and only a few newly added parameters are trained. Those parameters allow the LLM to learn new skills. In recent research, PEFT is also used to incorporate a new modality into the LLM. These parameters function as the bridge that learns to adapt the new modality. This also avoids the prohibitive computational and financial cost of fully retraining foundational LLMs.

LLaMA-Adapter [83] uses the PEFT method by introducing only 1.2 million trainable parameters to equip an LLM with visual reasoning capabilities. Its core innovation is a special zero-initialized attention mechanism designed to protect the LLM’s existing knowledge. This mechanism works by inserting new, trainable vectors, called adoption

prompts, into the LLM’s higher transformer layers. These prompts are intended to learn the new visual information. However, adding new, randomly initialized parameters can introduce noise that disrupts the LLM’s pre-trained knowledge. Zhang et al. solve this with a learnable gating factor. The gating factor is a single parameter that controls the influence of the adoption prompts. This gating factor is zero-initialized, so the new prompts have no influence on the LLM at the start of training, which allows training to be stable at the beginning. As training progresses, the gating factor slowly learns to increase, gradually “opening the gate” and allowing the now-meaningful adoption prompts to contribute to the model’s capability to understand multimodal input. This leads to a stable and effective adaptation.

Zhang et al. [82] present in their paper, **PILL: Plug Into LLM with Adapter Expert and Attention Gate**, another technique based on PEFT. The presented architecture is designed to help LLMs handle multiple modalities more efficiently by reducing the risk of modality interference. Modality interference describes the condition where different types of modalities interfere with each other and by doing so degrade the original knowledge of the LLM. To solve this Zhang et al. introduce three key components: the Mixture-of-Modality-Adapter-Expert (MoMAE), the Modality-Attention-Gate (MAG), and the Swish-Gated Linear Unit-Adapter (SwiGLU-Adapter). The MoMAE component uses separate expert models to process data of different modalities, preventing cross-modality interference from the beginning. The MAG component acts as a gate to adaptively control the contribution of each non-text modality to the overall input to the LLM. This stabilizes the training and prevents non-textual modalities from interfering with the LLM’s knowledge base. The SwiGLU-Adapter is an improved version of the standard adapter design. The authors apply the SwiGLU gating mechanism [61] to the adapter design proposed by Houlsby et al. [32]. The SwiGLU gating mechanism is a combination of the Swish function [58] and the Gated Linear Unit (GLU) [20]. The training for PILL is done in two stages. The first stage is called pre-training and aligns the model’s visual and textual information by training the projection layers between the modality adapters and the LLM as well as the adapters inside the LLM. In the second stage the authors fine-tune all introduced modules like the MoMAE, MAG and the projection layers on downstream tasks. The LLM itself remains frozen. With this approach Zhang et al. reached an average accuracy of 91.23% on the ScienceQA [47] benchmark, placing it among state-of-the-art models like LLaVA [44] and LaVIN [48].

Focusing on efficiency and cost-effectiveness, **the LaViN model**, proposed in [48] introduces the Mixture-of-Modality Adaptation (MMA) module. This module consists

of a visual adapter with a down-projection, SwiGLU, and up-projection layer, as well as a routing mechanism. The routing mechanism is used to dynamically decide which computational path to use based on whether the input is text-only or multimodal. This preserves the LLM’s text-only abilities, as text-only inputs bypass the visual adapter, resulting in zero additional computational cost for language tasks. This approach allows combining any LLM with any visual encoder. This approach is also theoretically applicable to other modalities, but Luo et al. [48] only show the combination of an LLM with a visual encoder. However, adding other modalities would require the developer to reimplement parts of the system because some components, such as the visual adapter proposed in the paper, are developed specifically for vision. To create the LaVIN model and show the effectiveness of the proposed approach, Luo et al. combine their newly proposed MMA module with a LLaMa model [64] and the Vit-L/14 [22] image encoder. The authors report that LaVIN can be trained in just 1.4 hours on eight A100 GPUs with only 3.8 million trainable parameters, achieving performance comparable to models like LLaVA [44] on benchmarks such as ScienceQA while using significantly fewer resources.

In a different **PEFT-based method, Vision as LoRA (VoRA)** [68] integrates visual capabilities by adding Low-Rank Adaptation (LoRA) [33] layers directly into the LLM. This method removes the need for additional external encoders or connector layers, incorporating everything into a single transformer, which is visualized in Figure 3.1.

To inject visual knowledge, Wang et al. use pre-trained vision models as teachers to guide the training of the LoRA layers via block-wise distillation. This distillation process forces the LLM’s intermediate visual representations to align with the teacher’s features, block by block, accelerating training and reducing the need for massive training data. Furthermore, VoRA modifies the LLM’s attention mechanism for visual tokens. It replaces the standard causal attention mask, which only permits attending to previous tokens, with a bi-directional mask. This allows visual tokens (image patches) to attend to each other in all directions, better capturing spatial relationships. A significant advantage of this LoRA-based approach is its effect on inference. After the training, the low-rank LoRA matrices can be mathematically merged back into the original frozen weights, resulting in a unified model that incurs near-zero additional computational or memory overhead compared to the base LLM. Another advantage is that VoRA allows handling images of different sizes because of the LLM’s inherent ability to handle inputs of different sizes also apply to the vision input. VoRA also has limitations. The most significant, as mentioned by the authors, is that VoRA requires additional pre-training

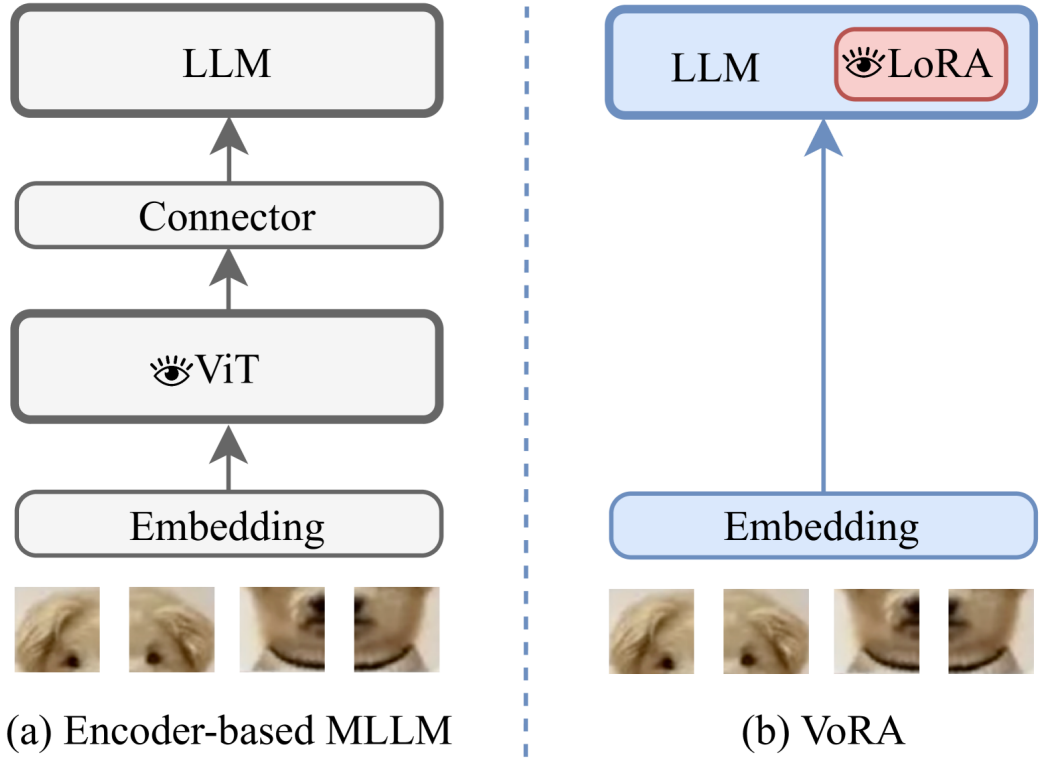


Figure 3.1: Architectural comparison between the encoder-based MLLM approach and VoRA [68]. The encoder-based architecture on the left routes image patches through a ViT and a connector before the LLM, whereas VoRA on the right replaces both components with LoRA layers inside the LLM.

data to compensate for the absence of external vision models, which are often used as experts for a specific modality in other approaches. Another limitation is the lack of a vision token compression mechanism. Unlike conventional MLLMs that use some kind of bridge module or projection layer, the authors intentionally preserved the original LLaVA-1.5 configuration to ensure fair comparison. Finally, the authors acknowledge that VoRA underperforms on world knowledge tasks, but claim that this is not due to architectural constraints but due to data limitations. Nevertheless, VoRA matches conventional encoder-based MLLMs in performance when trained with a proper scale of additional data.

While PEFT methods minimize the number of trainable parameters, they couple new modalities to the architecture of the LLM and thereby miss the opportunity to make use of pre-trained encoders, risking interference with the LLM’s knowledge base every

time a new modality is added. An alternative to this is shown through methods that use bridges to preserve the separation between modality-specific encoders and the LLM. These methods will be examined next.

3.2 Modular Architectures with Bridging Mechanisms

Another research trend involves modular designs that connect external pre-trained encoders to a frozen LLM. The key innovation in these architectures is the design of the bridge module that translates the encoder’s output into a format the LLM can understand. This bridge can range from a simple and lightweight design to a complex and more sophisticated one.

A notable method for integrating multimodal information into text-only LLMs is presented by [37] in their work on **FROMAGE**. The core of their multimodal understanding mechanism involves bridging the gap between a frozen visual encoder (ViT-L/14) and the frozen LLM (OPT-6.7B [84]). This connection is achieved not by fine-tuning the models themselves, but by training a lightweight linear projection layer as the bridge connecting the encoder with the LLM. This layer learns to project the visual embeddings produced by the image encoder into the LLM’s input embedding space.

In practice, when an image is provided as input, it is first encoded into a visual embedding. This embedding is then passed through the trained linear layer, transforming it into a vector that has the same dimensionality as the LLM’s text token embeddings. This new visual prefix is then prepended to the sequence of text embeddings. By treating the image representation as a prefix, the frozen LLM can condition its computations on the visual context without any modification to its architecture or weights. This has several benefits: specialized encoders with strong feature extraction capabilities can be used for each modality, and the separation of components makes the system easy to update and extend. Training the simple projection layers is also computationally less expensive than training complex bridging modules or fine-tuning large portions of an LLM, allowing the entire system to be trained on a single GPU in one day.

The projection layers are trained using an image captioning objective, optimizing the model to generate the correct text description given the visual prefix. The authors demonstrate that this adaptation allows the LLM to retain its in-context learning abilities, applying them to complex, visually-grounded tasks. While the model also introduces

a mechanism for producing multimodal outputs via image retrieval, it is not relevant to this work.

In [18] Cheng et al. used a hybrid architecture to implement **Emotion-LLaMA**. They adopted a straightforward architecture by combining distinct, pre-trained encoders for each modality (e.g., audio, video, text) with an LLM via simple linear projection layers, which are trained to translate the modality-specific features into the LLM’s shared embedding space, like in [37]. However, the LLM is not fully frozen. By using LoRA to tune the LLM during the instruction tuning phase, they also incorporated PEFT into their model. This technique of combining the modular encoder and the LLMs via simple projection layers and adding LoRA layers to the LLM makes Emotion-LLaMA especially effective in its specific task of emotion recognition.

In contrast to the simple projection layer method in [37] and [18], **mPLUG-Owl** [77] is a prominent example of a complex bridge. It consists of three main components: a foundational LLM, a so-called visual knowledge module, and a visual abstractor module. The foundational LLM, in this case LLaMA-7B [64], is the core of the system. It takes the text and visual data to compute a meaningful response for the user. The visual knowledge module is a pre-trained vision encoder, in this case the ViT-L/14 [22], that is used to process raw image data and turn it into embeddings. The visual abstractor module is an intermediate component between the visual knowledge module and the foundational LLM that addresses the sequence length problem: a high-resolution image processed by a ViT can produce hundreds or even thousands of embedding tokens, which would overwhelm the LLM’s context window. The abstractor compresses the numerous output embeddings from the vision encoder into a small, fixed set of learnable tokens. These condensed tokens represent higher-level semantic information and are then fed to the LLM.

Another main contribution of this paper is the two-stage training process, which consists of pre-training and instruction tuning. In the first stage, the pre-training, the model learns to align visual and textual data. Here, the visual knowledge module and the visual abstractor module are trained. To speed up the process of training the visual knowledge module the authors initialize it with the weights of a pre-trained visual encoder. The LLM stays frozen at this stage. In the second stage, the instruction tuning, the visual knowledge module stays frozen. The LLM and the visual abstractor are jointly fine-tuned, by using LoRA to fine tune the LLM. This fine-tuning is done with a joint dataset of both text-only and multimodal instructions. This two-stage method of

first aligning the visual components to a static LLM and then fine-tuning the LLM for multimodal instructions allows the model to achieve strong performance on multimodal benchmarks, such as the author’s proposed OwlEval, while maintaining its original linguistic capabilities. To assess the capabilities of the model the researchers created a new evaluation benchmark. Experimental results show that mPLUG-Owl outperforms existing models like MiniGPT-4 [86] and LLaVA.

mPLUG-Owl2 [76] builds on this architecture using LLaMA-2-7B [65] to address modality interference, like in [82], and enable modality collaboration, where different modalities enhance rather than degrade performance. The key innovation is the Modality-Adaptive Module (MAM). Instead of projecting visual features directly into the LLM’s shared embedding space, MAM decouples vision and language representations within the self-attention mechanism itself. It achieves this by using separate linear projection layers for the key and value matrices while sharing the query matrix. This design allows the model to learn relationships between modalities in a shared attention space while preserving the unique characteristics of each, mitigating interference. This collaboration is so effective that the authors report mPLUG-Owl2 excels at multimodal tasks and also shows significant improvement on text-only benchmarks. mPLUG-Owl2 matches or outperforms other state-of-the-art models in several benchmarks, including ScienceQA, MMBench [46] and MMLU [30]. The model also demonstrates zero-shot generalization to video question answering, achieving state-of-the-art performance on datasets like MSRVTT-QA [74].

Another similar method, which positions itself between the methods discussed in Section 3.1 and Section 3.2, is introduced by Huang et al. [35]. Here the LLM is modified not to understand a new modality but to save compute. The paper introduces **Multimodal Plug-and-Play-LLM (mPnP-LLM)** [35] a configurable lightweight LLM, especially designed for embodied AI. mPnP-LLM introduces an innovative method to reduce computational cost of MLLMs, by connecting the multimodal encoder to the last N layers of the LLM instead of the input layers. This gives users the possibility to configure how many layers of the LLM should be used, creating a trade-off between model performance, by using more layers and computational efficiency, by using fewer layers and thereby reducing the number parameters used in backpropagation and inference. To integrate the new modalities, the system uses trainable “Key-Value Aligners”, which are small Feed Forward Networks. The authors make use of the LLM’s attention mechanism, which operates on three types of vectors called: Queries (Q), Keys (K) and Values (V). The Key-Value Aligners are used to project the features of a modality en-

coder into new K-V pairs. These are then injected directly into the LLM’s self-attention mechanism, influencing the model’s processing at its higher, more semantic layers. The K, V projectors within the connected LLM blocks are fine-tuned using LoRA during adaptation.

These bridging methods decouple the encoders from the LLM but still require implementation work for each new modality, as a new bridge must be designed and trained every time.

3.3 Unified Embedding Spaces

A fundamentally different approach from the ones presented so far is the creation of a single, unified embedding space instead of a specific bridge between an LLM and one encoder. **ImageBind** [26] is a state-of-the-art example, learning a joint embedding space for six modalities: images, text, audio, depth, thermal, and Inertial Measurement Unit data. The core idea is to use images as a common link to bind all the modalities. This means instead of using data where every modality is paired with each other, ImageBind uses data where each modality is paired with image data. This solves a data collection problem: instead of needing $N \times N$ paired datasets (e.g., text-to-audio, audio-to-depth), the system only needs N datasets, each paired with images. By training separate encoders to align their outputs with image embeddings, ImageBind enables novel, zero-shot cross-modal tasks and emergent alignment between modalities not seen together during training. For instance, the authors fed audio embeddings from ImageBind into a text-to-image model, which then generated an image of a dog — a task it was never trained to do. Adding a new modality requires creating a specific encoder for it, gathering image-paired data for that modality, and training the new encoder to align its output embeddings to the ImageBind space.

The unified embedding space approach eliminates per-modality bridge implementations, but adding a new modality still requires code changes and a dataset that pairs the new modality with a predefined one (images, in the case of ImageBind).

3.4 LLM-as-Controller Frameworks

Finally, there is a separate class of multimodal systems that uses the LLM as a central controller that orchestrates other specialized models, rather than using the LLM in an end-to-end system.

HuggingGPT [62] is a prime example of this framework. It uses an LLM to manage a library of expert models, such as those available on Hugging Face, to solve complex, multi-stage tasks. The system operates in four stages:

1. **Task Planning:** The LLM analyzes the user’s request and breaks it down into solvable sub-tasks with dependencies on one another, which creates a clear roadmap of actions.
2. **Model Selection:** For each task, the LLM selects an appropriate expert model based on its text description available on Hugging Face. To manage the vast number of models, it first filters candidates by task, then ranks them by number of downloads. The result of the top-K models is then given to the LLM to choose from.
3. **Task Execution:** The selected models are invoked by the LLM, and their inference results are sent back to the LLM, which can then be used to either build a response to the user query or give the results to the next model as the input. The paper emphasizes the computational resource issue at this stage.
4. **Response Generation:** The LLM collects the results from all tasks and synthesizes them into a coherent, final response. Even if the response from one of the expert models is not text the response is brought into a textual format, like a file path if a file was generated by one of the models.

This framework makes it possible to combine any modalities and update the expert models without the need for retraining or implementing code. However, the paper also notes limitations dependent on the LLM’s capabilities, such as maximum context window length and the inherent planning ability of the chosen LLM.

A related framework by **Xie et al.** [73] focuses explicitly on a Training-Free orchestration method. This system also features a central LLM that calls expert models but adds a parallel text-to-speech architecture, that allows interaction by talking to the system, and a cross-modal memory system. This memory system stores interaction histories in JSON format, allowing the system to maintain context across different modalities and

improve conversational performance, aiming for a more integrated conversational assistant experience compared to the more task-execution-focused HuggingGPT. It contrasts with HuggingGPT by using a curated set of expert models rather than dynamically selecting from the entire Hugging Face repository.

This category represents a fundamentally different framework for building a multimodal system. By using the LLM as a controller or orchestrator of modality-specific models, the system can add new modalities at runtime and does not need to be retrained if a new modality is added. This avoids the retraining requirements of the other approaches but introduces new limitations, such as the context window and the planning ability of the orchestrating LLM. This approach may also reduce the representation ability of the modality-specific encoders because all modalities are converted to text, as language is used as the universal interface between all models. Another key limitation is the implementation of peripheral systems that are used to translate the output of the expert models into a human readable format.

3.5 Summary

This chapter has reviewed the evolving landscape of MLLMs, categorizing recent advancements by their core integration approach. There are two dominant approaches to integrate modalities into the LLM’s embedding space, which are accompanied by two additional approaches that follow their own integration method. The integration is done either through **PEFT methods** (see Section 3.1) that insert trainable modules directly into the LLM, or through **modular bridging methods** (see Section 3.2), where the bridging mechanisms range from simple linear layers to complex abstractors that connect external encoders to frozen or partially-tuned LLMs. While these approaches achieve state-of-the-art performance on established benchmarks, they remain dependent on implementing new bridge modules and retraining when introducing a novel modality.

Alternative approaches attempt to mitigate this rigidity but introduce their own limitations. **Unified embedding spaces** like ImageBind pre-align modalities but still require specific paired datasets and encoder training for extensions (see Section 3.3). On the other side **LLM-as-Controller frameworks** offer maximum flexibility by orchestrating expert models without retraining, but they are bottlenecked by the LLM’s planning capabilities, context window and the potential loss of information when converting rich multimodal outputs into text for the controller (see Section 3.4).

As noted throughout this review, nearly all current methods require explicit code changes to add new modalities to a MLLM.

4 System Design

The system consists of two primary components, the MLLM Designer and the Engine, whose interaction is shown in Figure 4.1.

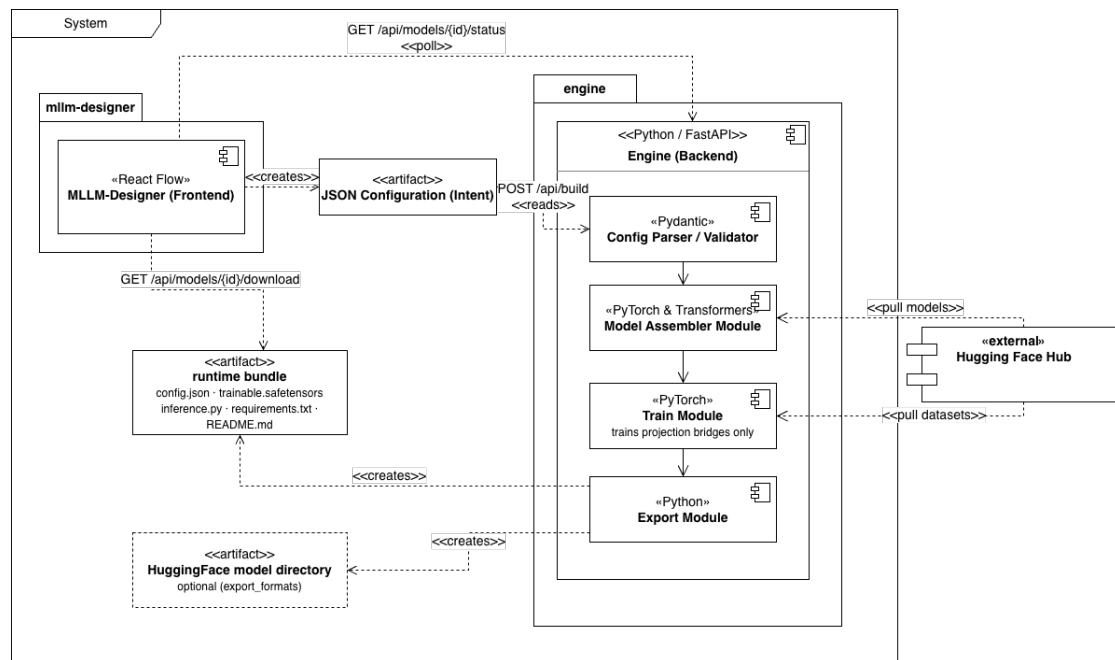


Figure 4.1: Overview of the system architecture with its two components, the MLLM Designer and the Engine. The Designer serializes the visual graph into a JSON intent and sends it to the Engine through a REST API. The Engine validates the intent, assembles the model, trains it, and exports the result as a runtime bundle or Hugging Face model directory.

4.1 Architectural Paradigm: Intent-Based Architecture

To address the problem stated in Section 1.1, the system adopts an Intent-Based Architecture. This approach aligns with concepts found in IBS and IBN (see Section 2.3).

The system architecture maps to the Intent Fulfillment (see Section 2.3) phase as follows:

1. **User Space:** The MLLM Designer acts as the user interface (Intent Interface) where the user defines “what” the system should look like. When a design is finalized, it is exported as a JSON configuration, which is a declarative description of the system. This artifact represents the desired State of the system.
2. **Translation Engine:** The Engine functions as the translation and execution mechanism. It parses the intent (the JSON) and determines the imperative steps (“how”) to instantiate the specific modules required to realize the design. After determining the steps, the Engine executes the built model.

4.2 The MLLM Designer

The first component is the MLLM Designer, shown in Figure 4.2.

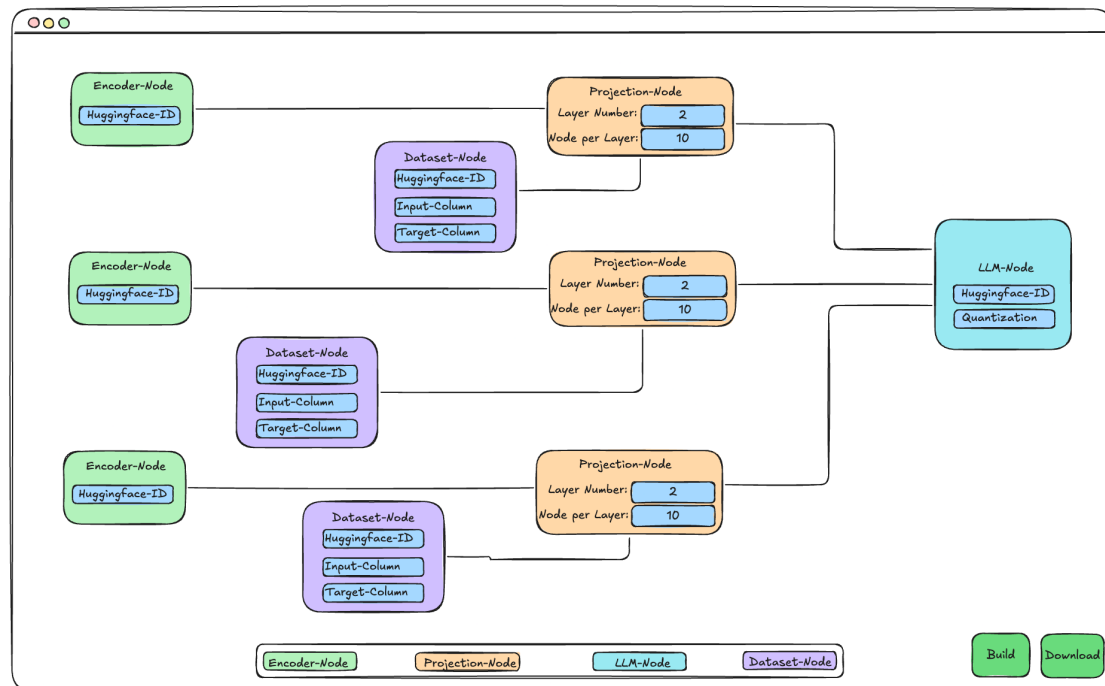


Figure 4.2: Wireframe of the MLLM Designer interface. The canvas shows encoder, projection, dataset, and LLM nodes connected as a directed graph, with a node palette at the bottom and build controls on the right.

It serves as the visual composition platform. It is implemented using **React Flow** [4], an open-source JavaScript library for building interactive node-based editors within React [2] applications. React Flow was selected because it is unopinionated, giving the developer flexibility in designing the components of the visual compositor. Because nodes are React components, the library does not impose strict constraints on the user interface. It natively provides canvas interactions such as panning, zooming, and drag-and-drop, so that the development focus can remain on domain-specific logic.

The MLLM-Designer allows users to build MLLM architectures via a drag-and-drop interface. The architecture is visualized as a directed graph where data flows from modality encoders through projection layers into a central LLM (see Section 3.2). The system defines four specific types of nodes:

- **Encoder Node:** This node holds the modality-specific encoders, which represent the sensory inputs of the multimodal system. The user configures this node by providing a string identifier corresponding to a pre-trained model in the Hugging Face repository and the modality that the provided encoder is able to handle.
- **LLM Node:** This node holds the LLM, which acts as the reasoning core of the system. Similar to the encoder, it accepts a string identifier for a specific language model available on Hugging Face. The node additionally exposes an optional 4-bit quantized loading path through the `load_in_4bit` field and a maximum sequence length.
- **Projection Node:** This node holds the projection layer that connects a modality encoder to the LLM. It contains an MLP that aligns the embedding spaces of the two models. The input dimension of this node is defined by the output dimension of the connected encoder, while its output dimension is defined by the input embedding size of the connected LLM. The user configures the internal architecture, specifically the number and size of hidden layers within the MLP.
- **Dataset Node:** This node represents the dataset used to train a specific projection layer. The dataset node expects a dataset ID from the Hugging Face repository together with a column mapping that assigns each input column to a modality and names the column that serves as the target or label for the training process. The modality assigned to an input column determines the preprocessing applied by the Engine and must match the modality of the connected encoder. The architecture treats data as a first-class component on an equal footing with model components. By decoupling the dataset definition from the projection layer, the system supports workflows such as training multiple adapters on the same data.

After the user has finished constructing the MLLM, they can start the build process by clicking the “Build” button. This serializes the visual graph into a JSON string (see Listing 4.1) and submits it to the Engine, which constructs the model and trains its projection layers. A “Check Status” button polls the state of the running build, and once the Engine reports the build as completed, a “Download Model” button appears through which the trained model is retrieved. Alternatively, the “Download JSON” button exports the serialized configuration itself, so that an intent can be stored, versioned, or shared without triggering a build.

The realized interface is shown in Figure 4.3 and follows the layout proposed in Figure 4.2.

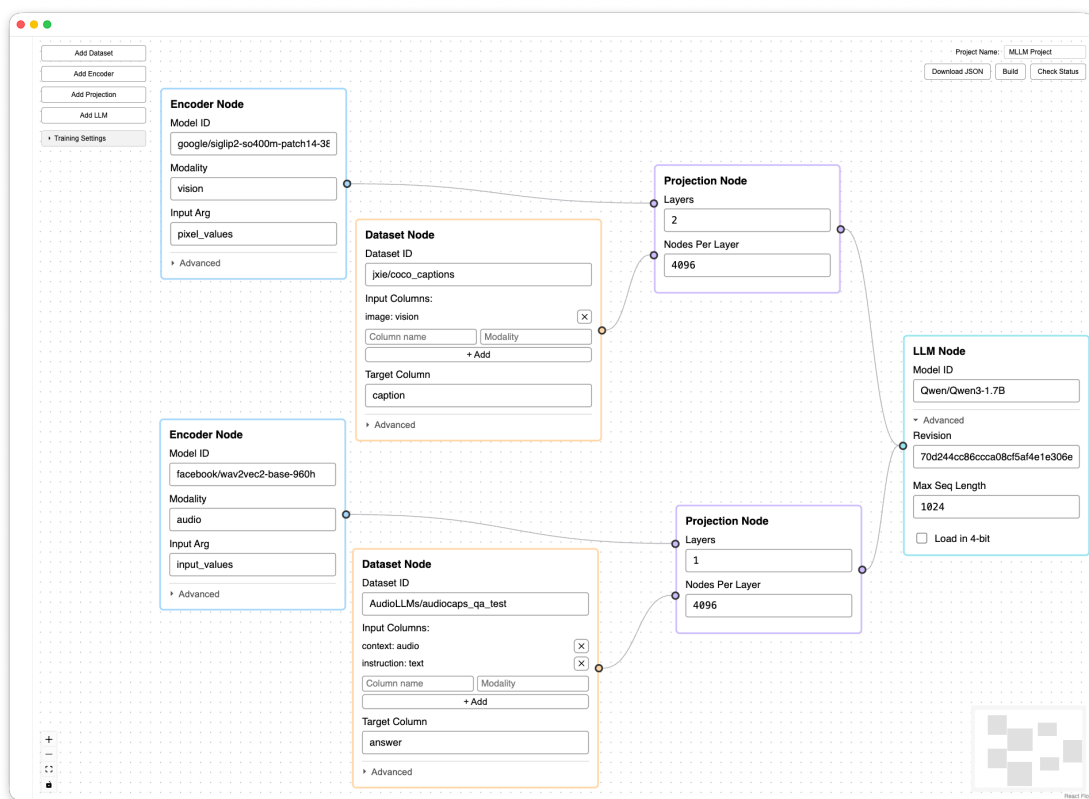


Figure 4.3: The implemented MLLM Designer with a two-modality pipeline on the canvas. A vision encoder and an audio encoder each connect through a dedicated projection node to a shared LLM, with dataset nodes supplying the training data for each projection layer.

The node palette occupies the left margin, while the project name field and the build controls described above are grouped in the upper right. Below the palette,

a collapsible “Training Settings” panel exposes the run-level parameters that are not tied to any single node and that form the `training_config` section of the exported intent (see Listing 4.1). The canvas holds a two-modality configuration in which a `google/siglip2-so400m-patch14-384` vision encoder and a `facebook/wav2vec2-base-960h` audio encoder each connect to a dedicated projection node before both projection layers converge on a `Qwen/Qwen3-1.7B` backbone. The vision projection layer draws its training data from `jxie/coco_captions`, while the audio projection layer is trained on `AudioLLMs/audiocaps_qa_test`, whose column mapping assigns the `context` column to the audio modality and the `instruction` column to text. Settings that control reproducibility and loading behavior, such as the `revision` pin, the `preprocessor_kind`, `max_seq_length`, and `load_in_4bit`, are placed behind a collapsible “Advanced” section on the encoder, dataset, and LLM nodes, so that the canvas remains readable while the full schema stays accessible. The LLM node in the figure shows this section expanded, exposing the pinned `revision`, the maximum sequence length, and the `load_in_4bit` option.

The exported JSON configuration serves as the system’s intent (see Section 2.3), encapsulating the MLLM’s blueprint, including model identifiers and projection layer definitions, without containing model weights or execution logic. The schema is organized into three sections: `project_meta` for identification and versioning, `training_config` for run-level parameters such as epochs, batch size, and learning rate, and `nodes` for the topological definition. Within the `nodes` section, the system employs a dependency-driven structure where downstream components, such as the LLM backbone, explicitly reference their upstream dependencies, specifically the projection layers, thereby modeling the MLLM as a directed acyclic graph. This design enforces a strict separation of concerns by isolating architectural decisions through parameters like `modality` that dictate preprocessing logic. Dataset, encoder, and LLM nodes may additionally pin a `revision`, pinning the referenced Hugging Face artifact to a specific commit so that a build remains reproducible even when the upstream repository changes. This schema ensures the constructed MLLM is modular, reproducible, and extensible. Listing 4.1 shows a truncated version of the configuration exported from the design in Figure 4.3, reduced to the vision modality path so that one node of each type is visible. The complete two-modality intent is reproduced in Listing A.1 in the appendix.

```

1 {
2   "project_meta": {
3     "id": "thesis-demo-project-001",
4     "name": "MLLM_Project",
5     "created_at": "2025-11-27T14:00:00Z"
6   },
7   "training_config": {
8     "trust_remote_code": false,
9     "export_formats": ["runtime"],
10    "max_seq_length": 2048,
11    "epochs": 2,
12    "batch_size": 16,
13    "accumulation_steps": 2,
14    "seed": 42,
15    "use_subprocess": false
16  },
17  "nodes": [
18    {
19      "id": "dataset_vision_1",
20      "type": "dataset",
21      "dataset_id": "jxie/coco_captions",
22      "column_mapping": {
23        "input_columns": { "image": "vision" },
24        "target_column": "caption"
25      },
26      "revision": null
27    },
28    {
29      "id": "encoder_vision_1",
30      "type": "encoder",
31      "model_id": "google/siglip2-so400m-patch14-384",
32      "modality": "vision",
33      "input_arg": "pixel_values",
34      "revision": null
35    },
36    {
37      "id": "bridge_vision_1",
38      "type": "projection",
39      "architecture": { "layers": 2, "nodes_per_layer": 4096 },
40      "inputs": {
41        "source_model": "encoder_vision_1",
42        "training_data": "dataset_vision_1"
43      }
44    },
45    {
46      "id": "llm_1",
47      "type": "llm",
48      "model_id": "Qwen/Qwen3-1.7B",
49      "load_in_4bit": false,
50      "max_seq_length": 1024,
51      "inputs": { "connected_bridges": ["bridge_vision_1"] },
52      "revision": "70d244cc86cca08cf5af4e1e306ecf908b1ad5e"
53    }
54  ]
55 }

```

Listing 4.1: Simplified JSON configuration exported from the MLLM Designer, showing a vision path.

4.3 The Engine

The second component is the Engine, a backend system written in Python. Designed as a RESTful service, it exposes endpoints to accept the serialized JSON intent from the Designer.

As soon as the Engine receives the JSON configuration, it parses the structural definition and translates the intent into executable code. For this translation, the Engine adopts a data-oriented design strategy by making use of discriminated unions (also known as typed enums) and structural pattern matching. Rather than relying on a traditional object-oriented factory pattern, which scatters validation and construction logic across multiple class hierarchies, the system centralizes the JSON interpretation logic and parses the JSON configuration into strictly typed data structures using the Pydantic [3] library. The `modality` and `type` fields act as discriminators, allowing the system to enforce schema validation and type-specific behavior at runtime. The Engine then employs structural pattern matching over these validated configurations to instantiate the requested components using the PyTorch framework [1] and the Transformers library from Hugging Face [72]. For LLM nodes, the Unsloth library [5] serves as an accelerated model loader with an optional 4-bit quantized loading path, exposed through the `load_in_4bit` field of the configuration. The Engine also initializes the corresponding modality-specific preprocessors (such as image processors or audio feature extractors) to ensure the training data matches the encoder’s expected input format.

The projection layers are initialized with random weights according to the requested architectural dimensions, representing the primary trainable parameters within this architecture. The Engine then executes an alignment training stage using the specified modality datasets, in which the projection layer weights are optimized to map the encoder outputs into the LLM’s text embedding space. When a configuration declares several modalities, the corresponding projection layers are trained sequentially. Each projection layer is optimized on the dataset connected to its own branch, while the input slots of all other modalities are filled by learned missing-modality embeddings that stand in for absent inputs. This design is motivated primarily by the availability of training data, because each projection layer is aligned on a single-modality corpus, no paired multimodal dataset is required, and any captioned corpus of the respective modality published on Hugging Face can serve as alignment data for its branch. It additionally keeps the training procedure simple, as each branch is optimized as an independent unit with its own dataset, so no joint sampling or scheduling strategy across modality corpora

is needed. A joint training stage over independently sourced corpora would moreover be ill-defined at the data level. Merging two single-modality datasets provides no correspondence between their samples, so there is no criterion for which audio clip should accompany which image, nor for which of the two datasets' captions would constitute the ground-truth target of such a combined sample. Joint multimodal training therefore presupposes a natively paired multimodal corpus, which the sequential per-branch alignment deliberately avoids requiring. As a consequence, no projection layer is ever optimized in a context where another modality slot holds a real input rather than its missing-modality embedding. A property whose implications for multimodal inference are examined in Section 5.

Upon completion of the training phase, the system intentionally avoids returning bare weight files, which would require the user to reconstruct the model graph by hand, as well as environment-dependent pickle files. Instead, the Engine produces a runtime bundle that is always emitted, complemented by an optional Hugging Face model directory that the user can request through the `export_formats` field of the training configuration.

The runtime bundle packages the JSON Intent originally submitted by the Designer, the trainable parameters serialized as a `trainable.safetensors` file, a standalone `inference.py` runtime, a `requirements.txt`, and a `README.md`. The trainable parameters consist of the projection layer weights together with the learned missing-modality embeddings introduced above. Because the frozen-encoder invariant guarantees that only these parameters participate in optimization, serializing them alone captures the entirety of the learned state. The frozen encoders and LLM are rematerialized at load time from their upstream Hugging Face identifiers recorded in the configuration. The `inference.py` runtime reads the JSON configuration, reconstructs encoders and projection layers according to the declared node graph, loads the trained weights, and exposes a command-line interface for multimodal generation, thereby giving the user a directly executable artifact rather than a collection of weights requiring glue code.

The optional `hf_model` export additionally materializes the full model, including frozen encoder and LLM weights, into a Hugging Face-compatible directory containing a `configuration_mllm.py` and `modeling_mllm.py` alongside the serialized parameters and tokenizer files.

Both artifacts are written to the project's output directory and bundled into a single zip archive that is returned to the frontend through the

`/api/models/{projectId}/download` endpoint. This export strategy decouples the generated model from the Engine’s internal Python class hierarchies and ensures that the resulting MLLM can be deployed without requiring the user to manage, maintain, or modify any underlying execution code, thereby fully realizing the system’s intent-based objective. The approach is motivated by the dynamic nature of the projection layers, whose architecture, including the number of hidden layers and their widths, is determined at runtime from the JSON configuration. Standard serialization formats such as ONNX and TorchScript require statically known tensor shapes at trace time and therefore cannot export projection layers that are initialized lazily from a user-defined specification.

5 Experiments

To test the functionality of the presented system several experiments were considered. The experiments are designed to test the primary use cases the system addresses, specifically the replacement of old components of the MLLM like an encoder with new ones and the expansion of a MLLM with a new modality by adding a new encoder to the system. The system was also used to reproduce an existing MLLM and evaluate its performance on the same benchmarks used in the original work. This chapter details the experiments conducted and the motivation behind them.

5.1 Experimental Setup

Experiments 1 and 2 are executed on a single NVIDIA A100-SXM4-40G GPU with 40 GB of VRAM, which also constitutes the minimum hardware requirement for running the system. Experiment 3 is executed on a single NVIDIA H100 NVL GPU with 94 GB of VRAM, due to the availability of compute resources at the time the experiment was conducted. The larger memory budget of this GPU also comfortably accommodates the reproduced LLaVA-1.5 setup, which pairs a 7-billion-parameter LLM with an input sequence length of 2048 tokens. Both servers run Ubuntu 24.04 LTS with Python 3.12, and all training and evaluation is executed inside the same Docker image with PyTorch 2.9.1 as the underlying runtime, so that the software stack is identical across hosts and only the GPU differs. The change of hardware does not affect the controlled comparison of Experiment 1, which is conducted entirely on the A100, and Experiment 3 is assessed primarily against the published LLaVA-1.5 reference scores rather than against the models of the other experiments. To ensure comparability of results across experiments, the same base LLM, namely `Qwen/Qwen3-1.7B`, is used in every configuration, and the training configuration is kept consistent. The training configuration is summarized in Table 5.1. All generated models are configured with an input sequence length

of 1024 tokens, with the exception of Experiment 3, which uses 2048 tokens to match the reproduced LLaVA-1.5 setup.

Parameter	Value
epochs	1
batch size	8
accumulation steps	4
validation ratio	0.1
optimizer	AdamW
learning rate	1e-3
max. optimizer steps	2000

Table 5.1: Training configuration used across all experiments. The step cap applies to Experiments 1 and 2 only.

The choice of Qwen/Qwen3-1.7B is motivated by the constraints imposed by the available hardware, as larger LLMs would exceed the VRAM budget once the encoder, projection layers, and activations of the modality data are taken into account. The batch size of 8, combined with four gradient-accumulation steps, follows the same reasoning. It yields an effective batch size of 32, which is large enough to stabilize the gradient updates of the projection layer, while the small per-device batch keeps the peak memory of a single forward pass within the VRAM budget. The per-device batch is the binding constraint here, since a single vision sample expands into 576 encoder tokens and the corresponding activations dominate the memory footprint of the step [45]. Training in Experiments 1 and 2 is additionally capped at 2000 optimizer steps (`max_steps` in the training configuration) rather than running a full pass over the training corpus. This cap serves as a fixed computational budget. Every configuration and every seed receives exactly the same number of gradient updates over the same sequence of training samples, so that an observed difference between two configurations cannot stem from unequal training exposure. The value of 2000 steps is motivated by preliminary runs, in which the training loss of the projection layer plateaued within the first few hundred optimizer steps. The cap therefore lies well beyond the point of diminishing returns while reducing the training time per run by roughly an order of magnitude compared to a full pass over the corpus. The cap also provides the learning rate schedule with a fixed horizon. The learning rate is warmed up linearly over the first 100 optimizer steps and then

follows a cosine decay that reaches zero at the end of the step budget, so that training concludes at a low learning rate rather than being cut off mid-schedule. Experiment 3 is excluded from this cap and trains for one full epoch, since it reproduces the Stage-1 procedure of LLaVA-1.5, which prescribes a complete pass over the alignment corpus [45]. In that case the decay horizon of the learning rate schedule is derived from the corpus size instead. Throughout all experiments, only the projection layers are trained while both the modality encoders and the LLM remain frozen. This is consistent with the architectural assumption laid out in Section 4.

Experiments 1 and 2 use the COCO Captions corpus [16] as the training dataset for the vision projection layer. COCO Captions is a widely used image-caption corpus with broad object and scene coverage, and is a standard reference for feature-alignment training in the MLLM literature [29, 44]. It is held constant across both experiments so that any observed difference between configurations can be attributed to the architectural change under test rather than to the training data. Experiment 3 uses a different dataset (`liuhaotian/LLaVA-Pretrain`) because that experiment reproduces a published model and must match the original training corpus.

To support reproducibility, every configuration is trained under three fixed random seeds (42, 2, and 534), each seeding PyTorch, NumPy, and Python’s built-in random module, with CUDA deterministic mode enabled. Running each configuration across multiple seeds separates the effect of the architectural change under test from run-to-run variation caused by random initialization and data ordering, so that reported differences can be attributed to the configuration rather than to a single fortunate or unfortunate seed. Table 5.2 lists the exact model and dataset identifiers, each pinned to a specific Hugging Face commit, for all components used across the three experiments. Pinning the revision together with the fixed set of seeds makes each run reproducible. The frozen encoder and LLM weights, and the training and evaluation data, are identical regardless of when the run downloads them. The commits below correspond to each repository’s main branch as of 2026-07-10 and are enforced in code via the `HF_REVISIONS` registry in `engine/experiments/_common.py`. For readability the revisions are abbreviated to their first four characters here, while the registry records each of them in full.

Component	Hugging Face Identifier	Rev.
LLM (Exp. 1–2)	Qwen/Qwen3-1.7B	70d2
LLM (Exp. 3)	lmsys/vicuna-7b-v1.5	3321
Vision Encoder (Exp. 1: baseline)	openai/clip-vit-large-patch14-336	ce19
Vision Encoder (Exp. 1: swap)	google/siglip2-so400m-patch14-384	e8e4
Audio Encoder (Exp. 2)	facebook/wav2vec2-base-960h	22aa
Train data (Exp. 1–2)	jxie/coco_captions	a2ed
Train data (Exp. 2: audio)	AudioLLMs/audiocaps_qa_test	6cf6
Train data (Exp. 3)	liuhaotian/LLaVA-Pretrain	70f9
Eval: MMBench-EN (1, 3)	lmms-lab/MMBench_EN	ecd2
Eval: ScienceQA (Exp. 3)	derek-thomas/ScienceQA	f18b
Eval: MSR-VTT (Exp. 2)	mteb/MSR-VTT	4661

Table 5.2: Model and dataset identifiers, pinned to Hugging Face commits, used across all experiments.

For reproducibility, the experiments are not executed interactively through the MLLM Designer, but by scripts located in `engine/experiments/`. The scripts programmatically construct the same JSON intent configuration that the MLLM Designer serializes (see Section 4) and hands it to the unmodified Engine. This choice is motivated by requirements that concern the orchestration around a build rather than the build itself, such as repeating each configuration across the three fixed seeds, sourcing the revision pins of Table 5.2 from a single central registry shared by all runs, and running the benchmark evaluations automatically after each training run. The Designer therefore remains the authoring interface the system offers to users, while the experiment scripts serve as its scripted equivalent for controlled experimentation. The configurations the scripts use conform to the same intent schema the Designer exports, extended only by the reproducibility metadata named above. In addition to constructing the configurations, the experiment scripts prepare the training corpora by reshaping them into the input format consumed by the Engine’s dataset nodes. This preparation is data plumbing rather than a modification of the Engine or of any model code, and it therefore does not affect the code-free claims tested in the following experiments.

5.1.1 Training Within Hardware Constraints

Alongside the composition logic itself, a substantial share of the engineering effort went into making training feasible on the available hardware. Even with frozen backbones, training an MLLM is memory-intensive, as the LLM, one or more encoders, their activations, and the optimizer state of the projection layers must coexist within the 40 GB of VRAM of the minimum hardware configuration named above.

The frozen-backbone invariant is therefore exploited systematically rather than treated as a mere training convention. Because the encoders and the LLM are excluded from optimization, the AdamW optimizer is constructed only over the trainable projection parameters, no gradient state is allocated for the frozen components, and encoder forward passes run without gradient tracking. The same principle extends to persistence, where the best-model snapshot and the safetensors export retain only the trainable weights, keeping checkpoints orders of magnitude smaller than the full model graph. Peak memory during training is further bounded by gradient accumulation [54] and by gradient checkpointing of the frozen LLM, which trades recomputation time for activation memory. On the data side, datasets are consumed as Hugging Face `IterableDataset` streams rather than downloaded in full, and each training corpus is additionally materialized once into a local cache keyed on its pinned dataset revision so that repeated runs do not need to re-download the source data.

The handling of numerical precision underwent a design iteration worth recording. An earlier version of the trainer used fp16 autocast together with a gradient scaler, the standard mixed-precision recipe. A baseline run of Experiment 1 exposed the limits of this approach, as the gradient scaler protects gradients from underflow but does not constrain the forward pass, and activations exceeding fp16’s maximum representable value of approximately 65,504 rendered the loss itself NaN. Precision handling was consequently centralized in `engine/src/models/precision.py`, which selects bfloat16 autocast only on GPUs with native bfloat16 support and falls back to full fp32 without autocast otherwise, favoring numerical robustness over the memory savings of fp16.

Not every optimization survived into the final system. An earlier version of the Engine cached the embeddings of the frozen encoders on disk to avoid redundant forward passes. This cache was removed again when the complexity of cache warm-up and invalidation outweighed its observed benefit in the streaming-based pipeline. The shipped system is

in this sense the residue of a larger explored design space, reduced to the components whose benefit outlasted contact with the real training workloads.

Taken together, these measures made the experiments of this work executable on the single-GPU configurations described above. They managed the memory pressure rather than resolving it. The constraints that remain despite these measures are discussed in Section 7.1.

5.1.2 Scoring Protocols

The models produced in Experiments 1 and 3 are trained exclusively on caption-style alignment data and receive no instruction tuning. Preliminary runs showed that they respond to multiple-choice questions with caption-style prose from which no option letter can be parsed, and generation-based scores consequently collapse toward zero regardless of the quality of the learned feature alignment. This failure mode is not specific to the present system. The MMBench authors report that vision-language models answer in free-form text rather than with an option letter, and their official protocol therefore extracts answers from free-form output through rule-based matching with a ChatGPT-based fallback [46]. For this reason, every multiple-choice benchmark in this work, namely MMBench and ScienceQA, is scored under two protocols. The captioning evaluation of Experiment 2 is not affected by this failure mode and is scored with a captioning metric as described there.

Under the generation-based protocol, the model generates a short completion, and the answer is parsed from the output according to the parsing rule given in the respective benchmark description. Because unparseable outputs are counted as incorrect, this score primarily measures answer-format compliance for projection-only models.

Under the likelihood-based protocol, no text is generated. Instead, the model scores each candidate answer by computing the length-normalized log-probability of the candidate as a continuation of the question, following the procedure described in Section 2.1.4, and the candidate with the highest likelihood is taken as the prediction. This answer-ranking approach is the standard multiple-choice method of the lm-evaluation-harness framework [13] and was applied to visually conditioned language models by Flamingo [6]. The SEED-Bench MLLM benchmark adopted the same method explicitly so that scores do not depend on a model’s instruction-following ability to output an option letter [41]. For MMBench and ScienceQA, the ranked candidates are the textual contents of the answer

options. Because every question receives exactly one valid prediction under this protocol, the random-chance baseline is meaningful and scores can be interpreted against it. Wang et al. [71] show that likelihood-based and generation-based multiple-choice results can diverge substantially. Both protocols are therefore applied in the results sections of the respective experiments, though generation-based scores serve only to document the anticipated format-compliance failure, while the likelihood-based scores provide the basis for analysis. Scores obtained under different protocols are never compared against each other. Following their analysis, the ranking is computed over the full textual content of each option rather than over the probability of a single option-letter token.

5.2 Experiment 1: Component Replacement

5.2.1 Motivation and Objective

The first experiment is concerned with the modularity claim of the system: that an existing component of an MLLM can be replaced by a different one without any direct code changes, purely through the intent configuration described in Section 4. One way to test this property is to construct a baseline MLLM with an established but comparatively older encoder, evaluate it, and then build a second MLLM that differs only in the choice of encoder. Because the system does not support loading a previously built model, the “replacement” consists of creating a new intent configuration with the updated encoder, from which the second model is constructed from scratch.

5.2.2 Hypotheses

H1: A vision encoder replacement can be performed entirely through the JSON intent configuration, without any modifications to the Engine code base.

H2: Replacing the older CLIP ViT-L/14-336px encoder with the newer SigLIP-2 SO400M encoder, while holding the LLM and training setup constant, leads to improved scores on the MMBench benchmark.

5.2.3 Procedure

In the first step, a baseline MLLM is declared in the JSON intent configuration using `Qwen/Qwen3-1.7B` as the LLM and `openai/clip-vit-large-patch14-336` as the vision encoder, connected through a single projection layer. The configuration is handed to the Engine, which assembles the model and trains the projection layer on COCO Captions according to the configuration in Table 5.1. The trained baseline model is then evaluated on the MMBench benchmark, which serves as the reference score for the comparison.

In the second step, a new intent configuration is created that is identical to the baseline configuration except that the encoder node now references `google/siglip2-so400m-patch14-384` instead of `openai/clip-vit-large-patch14-336`. The LLM node, the projection node, and the dataset node remain unchanged. The new configuration is submitted to the Engine, which assembles the model from scratch and trains the projection layer. Because the input dimension of the projection node is derived from the connected encoder (see Section 4), the projection layer is automatically initialized with the dimensions required by the new encoder.

In the final step, the rebuilt model is evaluated on the same benchmarks as the baseline. The two scores are compared in order to assess whether the upgrade to the newer encoder yields a measurable performance improvement. The experiment is considered successful if both configurations are built and trained without any code changes to the Engine (H1). A measurable improvement in benchmark scores would additionally confirm H2 and demonstrate that the modularity offered by the system translates into tangible benefits for the resulting MLLM.

5.2.4 Benchmarks

MMBench [46] was selected because it is among the most widely adopted benchmarks for MLLM evaluation in the literature [78] and provides broad multiple-choice coverage of visual understanding abilities across 20 fine-grained dimensions organized into a three-level taxonomy. It evaluates models through multiple-choice VQA, where each question is associated with one specific ability. The official evaluation protocol uses CircularEval, a strategy in which each question is presented multiple times with permuted answer choices, to reduce the influence of position bias in the model’s predictions. In this work, a simpler single-pass evaluation on the `dev` split is used instead, with each

question posed once with its original option order. The benchmark is scored under two complementary protocols, a generation-based protocol and a likelihood-based protocol, which are described in Section 5.1.2. Under the generation-based protocol, the model generates a completion of at most four tokens, and the option letter parsed from the output is taken as the prediction. Outputs from which no option letter can be parsed are counted as incorrect. A single aggregate accuracy is reported. A per-ability breakdown is not produced.

5.2.5 Results

Both configurations were declared, assembled, and trained purely from their intent configurations, with the two configurations differing only in the referenced encoder node, and no changes to the Engine were required to build or train either model. This confirms H1: the encoder replacement is expressed entirely through the intent configuration.

The generation-based protocol confirmed the format-compliance failure anticipated in Section 5.1.2. A generation-based evaluation of the SigLIP-2 configuration under seed 2 yielded an MMBench accuracy of 0.003, as the caption-trained projection layer responds to the multiple-choice prompts with descriptive prose rather than an option letter. These scores are therefore not interpretable as a capability comparison, consistent with the rationale for the likelihood-based protocol. The likelihood-based MMBench scores are reported per seed in Table 5.3, together with the mean over the three seeds.

Configuration	Seed 2	Seed 42	Seed 534	Mean
CLIP (baseline)	0.345	0.345	0.332	0.341
SigLIP-2 (swap)	0.369	0.369	0.360	0.366

Table 5.3: Likelihood-based MMBench accuracy for Experiment 1 under seeds 2, 42, and 534. The baseline configuration uses `openai/clip-vit-large-patch14-336` and the swap configuration `google/siglip2-so400m-patch14-384`, with all user-specified settings held identical and the projection layer adapting automatically to the encoder’s output dimension. The measured chance level of the single-pass dev evaluation is 0.266.

Under the likelihood-based protocol, both configurations score above the measured chance level of 0.266 by a clear margin (means of 0.341 and 0.366), indicating that the projection layers did learn a usable feature alignment between the frozen encoder and the frozen LLM. To verify that this margin reflects visual information rather than the language prior of the LLM, both configurations were additionally scored under three control conditions, reported in Table 5.4 and discussed below.

The SigLIP-2 configuration outperforms the CLIP baseline with non-overlapping per-seed ranges. Every SigLIP-2 seed (0.360, 0.369, 0.369) scores above every CLIP seed (0.332, 0.345, 0.345), and the gap between the means is 0.025. H2 is therefore supported, as the newer encoder yields a consistent improvement that exceeds the seed-level variation observed in either configuration.

To determine whether the above-chance MMBench scores reflect learned visual alignment rather than the language prior of the frozen LLM, both configurations were re-evaluated under three control conditions on the same question set. Question-only baselines and mismatched-image ablations are established diagnostic techniques in VQA robustness research [27, 49], and Chen et al. [15] demonstrate their relevance for MLLM evaluation by showing that LLMs answer a substantial fraction of MMBench and ScienceQA questions correctly from the language prior alone. Under the text-only condition, the visual prefix is suppressed entirely, so the LLM answers from the question and answer options alone. Under the shuffled condition, each question receives another question’s image, preserving the presence of a visual prefix while removing the correct visual information. Under the random-projection condition, the real image is encoded but passed through a re-initialized projection whose weights are drawn from the same distribution used at the

start of training, isolating the effect of projection training from the effect of the image itself. Table 5.4 reports the mean over the three seeds for each condition, alongside the trained scores and the measured chance level.

Condition	CLIP (baseline)	SigLIP-2 (swap)
Trained	0.341	0.366
Shuffled	0.302	0.296
Text-only	0.287	0.287
Random projection	0.280	0.281
Chance	0.266	0.266

Table 5.4: Mean MMBench accuracy over seeds 2, 42, and 534 under four scoring conditions. Trained is the standard evaluation with the learned projection. Shuffled feeds another question’s image. Text-only suppresses the visual prefix entirely. Random projection passes the real image through a re-initialized projection. Chance is $\text{mean}(1/n_{\text{options}})$ over the evaluated questions. The text-only result is identical across all six bundles because suppressing the prefix collapses each bundle to its bare LLM.

The trained models exceed the text-only baseline by 5.4 percentage points for CLIP and 7.9 percentage points for SigLIP-2, confirming that the learned projections transmit visual information that the LLM exploits. The shuffled condition, which preserves the presence of a visual prefix but provides the wrong image, scores 3.9 and 7.0 percentage points below the trained condition, so the improvement is attributable to the correct image rather than to the mere presence of visual tokens. The random-projection condition sits just above chance (0.280 and 0.281 against 0.266), confirming that the untrained projection contributes negligibly and that the trained scores are a product of projection training. The language prior of the LLM itself accounts for only 2.1 percentage points above chance (text-only 0.287 against chance 0.266), which is small relative to the trained margin and confirms that MMBench is not dominated by text-answerable questions in this evaluation.

The controls also provide a second perspective on the encoder comparison. The SigLIP-2 configuration gains 7.9 percentage points over its own text-only baseline, compared to 5.4 for CLIP. This indicates that the newer encoder not only scores higher in absolute terms but also extracts more image-specific information through the projection layer.

5.3 Experiment 2: Multi-Modality Expansion

5.3.1 Motivation and Objective

While the first experiment tests the replacement of an existing component, the second experiment focuses on the complementary property of extensibility, or in other words, the ability to expand an existing MLLM by adding an entirely new modality. The system is designed to treat each modality as an independent branch consisting of an encoder, a projection layer, and a dataset (see Section 4), all connected to the same LLM node. Adding a new modality should therefore reduce to declaring the additional nodes in the intent configuration alongside the existing ones, without altering the Engine itself.

This experiment tests this property by starting from a vision-only MLLM and extending it with an audio modality, resulting in a Vision + Audio architecture.

5.3.2 Hypotheses

H3: Adding an audio modality branch to an existing vision-only MLLM is achievable by declaring additional nodes in the intent configuration, without any changes to the Engine’s execution logic.

H4: The resulting Vision + Audio model performs at least on par with the vision-only baseline on MSR-VTT (MSR Video to Text), where audio carries complementary information, indicating that the new modality is correctly aligned with the LLM’s embedding space.

5.3.3 Procedure

In the first step, a vision-only baseline MLLM is declared in the intent configuration with `google/siglip2-so400m-patch14-384` as the vision encoder, a single projection layer, and `Qwen/Qwen3-1.7B` as the LLM. This configuration is built and trained by the Engine on COCO Captions as described in Section 4, and its baseline performance is recorded on the MSR-VTT benchmark.

In the second step, a new intent configuration is created that reproduces the vision branch from the first step and additionally declares a second modality, namely an encoder node

referencing `facebook/wav2vec2-base-960h`, a new projection node, and a new dataset node referencing `AudioLLMs/audiocaps_qa_test` as the audio training data. This corpus is a Hugging Face redistribution of the test split of AudioCaps [36], a widely used dataset of audio clips annotated with natural-language descriptions of the audible events, and it therefore fills the same feature-alignment role for the audio projection layer that COCO Captions fills for the vision projection layer. The choice of this particular distribution is pragmatic, since the Engine loads training data directly from the Hugging Face Hub, and the original AudioCaps release provides only YouTube identifiers whose audio must be retrieved separately, whereas this distribution embeds the audio waveforms in the dataset itself. Although the corpus is published upstream as an evaluation set for audio-capable LLMs, using it for training introduces no train–test leakage, because this experiment is evaluated exclusively on MSR-VTT. With 313 clips, the resulting audio alignment corpus is considerably smaller than the vision corpus, a limitation that is taken up again in the interpretation of the results below. The configuration is handed to the Engine, which assembles and trains the complete Vision + Audio MLLM, including both projection layers, according to the configuration in Table 5.1. The resulting Vision + Audio model is then evaluated twice on MSR-VTT: once with vision-only inputs and once with both vision and audio inputs. Because both evaluations use the same trained model, the difference between the two BLEU-1 scores isolates the contribution of the audio input at inference time, while the comparison against the separately trained vision-only baseline captures the effect of the architectural extension as a whole.

The experiment is considered successful if the addition of the audio branch is achievable entirely through the intent configuration, without modifying the Engine (H3). If the resulting Vision + Audio model additionally performs at least on par with the vision-only baseline on MSR-VTT, this confirms H4 and provides evidence that the new modality is correctly aligned with the LLM’s embedding space.

5.3.4 Benchmarks

MSR-VTT was selected because it is one of the few widely used video description benchmarks whose clips retain their original audio channel alongside the visual content [75], making it inherently multimodal. Purely visual benchmarks such as MMBench would not be able to reveal the contribution of the audio branch, as they contain no audio signal. Audio-visual benchmarks such as DAVE [57], which require both modalities to

answer correctly, were considered but not selected because DAVE is designed as a diagnostic tool for evaluating audio-visual alignment in isolation, whereas this experiment needs a benchmark where a vision-only baseline can already achieve a score so that the incremental contribution of the audio branch becomes visible.

MSR-VTT comprises 10K web video clips with 200K clip-sentence pairs across 20 representative categories. Because many of its clips contain both meaningful visual content and an informative audio track, MSR-VTT provides a setting in which a vision-only model can already achieve a baseline through visual cues alone, while a model that additionally processes audio has access to complementary information that can improve its answers. This property makes MSR-VTT well suited for the purpose of this experiment. The vision-only baseline establishes a reference score, and an improvement observed after adding the audio branch can, under the evaluation protocol described below, be attributed to the contribution of the new modality.

The evaluation uses the `mteb/MSR-VTT` dataset, whose test split provides 879 clips with the media embedded directly in the dataset. A data sample consists of a video track, a separate 16 kHz audio track, and one reference caption per clip. This source is chosen specifically because its clips retain their original audio, a property that the widely mirrored `friedrichor/MSR-VTT` distribution does not preserve. Its clips are silent, which would make the audio branch indistinguishable from vision-only inference. Each clip is represented by its middle frame, which is passed to the vision encoder as a single representative image. This is a deliberate simplification as temporal dynamics across frames are not captured, so the benchmark serves here as a caption-quality proxy rather than a full video-understanding evaluation. When the audio branch is evaluated, the clip’s accompanying 16 kHz mono audio track is passed to the audio encoder. For each clip, the model generates a one-sentence description of at most 32 tokens. The reported metric is unigram BLEU (BLEU-1) [55] with a brevity penalty, computed against the clip’s single reference caption and averaged over all evaluated clips. While more elaborate captioning metrics exist, BLEU-1 suffices for the relative comparison at the core of this experiment, since all configurations are scored under an identical protocol.

5.3.5 Results

As in the first experiment, the Vision + Audio model was assembled and trained by the Engine directly from the extended intent configuration, which reproduces the vision

branch of the baseline and declares the audio encoder, projection, and dataset nodes alongside it. No changes to the Engine’s execution logic were required to build, train, or evaluate the extended model. This confirms H3 as the addition of a new modality branch is expressed entirely through the intent configuration.

The BLEU-1 scores of all three evaluation settings are reported per seed in Table 5.5, since the central comparison of this experiment is made within each trained Vision + Audio model rather than between averaged configurations.

Configuration	Seed 2	Seed 42	Seed 534	Mean
Vision-only baseline	0.1832	0.1790	0.1780	0.180
Vision + Audio, vision-only inference	0.1833	0.1802	0.1823	0.182
Vision + Audio, vision and audio inference	0.1006	0.1402	0.1512	0.131

Table 5.5: BLEU-1 scores on the `mteb`/MSR-VTT test split for Experiment 2 under seeds 2, 42, and 534. The two lower rows evaluate the identical trained Vision + Audio model, once restricted to vision inputs and once with the audio track additionally provided, so their per-seed difference isolates the contribution of the audio input at inference time.

The comparison between the first two rows captures the effect of the architectural extension on the existing modality. Under vision-only inference, the Vision + Audio model matches or slightly exceeds the separately trained vision-only baseline under every seed, and the per-seed spread of both configurations is narrow at 0.005. Declaring and training the additional audio branch therefore did not degrade the vision pathway, which is consistent with the architectural assumption that modality branches are trained and operate as independent units (see Section 4).

The contribution of the audio input at inference time, however, is negative. Providing the audio track lowers the BLEU-1 score under every seed, from 0.1833 to 0.1006 under seed 2, from 0.1802 to 0.1402 under seed 42, and from 0.1823 to 0.1512 under seed 534. This gap holds across all seeds. Even the weakest vision-only score of 0.1802 lies clearly above the strongest with-audio score of 0.1512. H4 is therefore not confirmed because the Vision + Audio model does not perform at least on par with its vision-only inference when the audio input is provided, so the audio branch, although functional, does not contribute complementary information under this evaluation.

This degradation is explicable from the training regime rather than from a defect of the extension mechanism. The Engine trains each projection layer sequentially on its own modality dataset, the vision projection layer on COCO Captions and the audio projection layer on AudioCaps, so no training sample ever contains both modalities, and while either branch is trained the other modality slot is filled by the learned missing-modality embedding (see Section 4.3). Joint vision-and-audio conditioning at inference therefore presents the frozen LLM with an input combination that was never observed during training. Two further factors increase this effect: the audio projection layer was aligned on curated audio-event descriptions from AudioCaps, whereas MSR-VTT soundtracks consist of in-the-wild web-video audio such as music, speech, and ambient noise, and the audio alignment corpus is considerably smaller than the vision corpus. The seed-level behavior points in the same direction, as the magnitude of the degradation varies strongly across seeds (0.031 to 0.083) while the vision-side scores vary by no more than 0.005, which is consistent with an undertrained and therefore unstable audio pathway. Dodge et al. [21] show that fine-tuning outcomes vary substantially across random seeds, and that this variance is amplified when the training set is small, which matches the pattern observed here for the audio branch aligned on only 313 samples. That an added modality can lower rather than raise performance is a known difficulty of multimodal training. Wang et al. [70] find that a late-fusion audio-visual classification network is outperformed by its vision-only counterpart, and attribute this to the modalities overfitting and generalizing at different rates under a single shared optimization strategy. That mechanism does not transfer directly to this experiment, whose branches are aligned separately rather than jointly. The same authors note, however, that architectures which avoid joint training by combining independently trained uni-modal components are themselves likely to be sub-optimal for that very reason, which is the situation the sequential per-branch alignment produces here.

In summary, the extensibility property under test is confirmed at the workflow and architecture level. The new modality is added purely declaratively (H3), and its addition leaves the existing modality unimpaired. The performance criterion of H4 is not met, as inference-time audio input lowers the caption quality instead of preserving or improving it. This outcome indicates that the sequential per-branch alignment executed by the Engine suffices to integrate a new modality mechanically, but not to calibrate the joint use of several modalities at inference, for which a jointly multimodal training stage would be required.

5.4 Experiment 3: Reproducing a Published Architecture

5.4.1 Motivation and Objective

The third experiment validates the system’s ability to reproduce an existing MLLM architecture entirely through the JSON intent configuration, without writing any custom model code. A successful reproduction demonstrates that the intent configuration and the Engine can express real-world architectures that have been independently designed and published, not only the configurations used in the previous two experiments, and that the resulting model is functional in the sense that it performs measurably above chance, even though the omission of the instruction tuning stage likely places it below the published results.

5.4.2 Hypotheses

H5: The system can express the full LLaVA-1.5 Stage-1 architecture: specifically CLIP ViT-L/14-336px encoder, two-layer MLP projection, and Vicuna-7B LLM, through the intent configuration, without any custom model code.

H6: The resulting model performs above random chance on ScienceQA (image subset) and MMBench under the likelihood-based scoring protocol (see Section 5.1.2), confirming that projection-only training produces a functional feature alignment between the vision encoder and the LLM.

5.4.3 Choice of Model

LLaVA-1.5 [45] was selected as the target model for the reproduction based on the following criteria. Architecturally, LLaVA-1.5 follows a clean encoder–projection–LLM pattern, in which a CLIP ViT-L/14-336px vision encoder is connected to an LLM through a two-layer MLP projection with Gaussian Error Linear Unit (GELU) activations. This pattern maps directly onto the component types supported by the system (see Section 4), and it does not require any additional mechanisms such as LoRA or other adapter modules, which are not currently supported. The training data required for the feature alignment stage is publicly available on Hugging Face, which is the only dataset source the system currently supports. `liuhaotian/LLaVA-Pretrain` provides the LCS-558K

image-text pair subset used for Stage 1 in LLaVA-1.5. The original 2023 LLaVA paper [44] used a filtered 595K subset of CC3M for Stage 1, but reproducing that exact corpus in 2026 is impractical due to heavy link rot in the underlying CC3M image URLs. LLaVA-1.5’s LCS-558K is the reproducible choice and is bundled with its images on Hugging Face. LLaVA-1.5 also reports results on well-established and publicly accessible benchmarks, namely ScienceQA and MMBench, which overlap with the benchmarks used in Experiment 1 and therefore allow the evaluation pipeline to be reused. Finally, LLaVA and LLaVA-1.5 are widely cited and reproduced MLLMs in the literature, with open-source code, weights, and detailed documentation of their training procedures, which makes them a robust target for a reproduction study [29, 81].

LLaVA-1.5 uses Vicuna-7B/13B v1.5 as its language backbone. Unlike Experiments 1 and 2, which use Qwen/Qwen3-1.7B across all configurations, this experiment uses `lmsys/vicuna-7b-v1.5` as the base LLM to stay as close to the original architecture as possible. This substitution also serves as an additional demonstration of the system’s modularity. Because the LLM node is simply exchanged in the intent configuration, and the Engine handles the change without any code modification.

LLaVA-1.5 employs a two-stage pipeline in which the second stage (visual instruction tuning) jointly fine-tunes the projection layer and the LLM. Because the system trains only projection layers while all backbones remain frozen (see Section 5), this experiment omits the second stage and trains only the feature alignment stage. The evaluation additionally uses the likelihood-based protocol rather than the generation-based protocol of the original paper (see Section 5.1.2), so the resulting scores are not directly comparable to the published figures. The comparison therefore focuses on whether the system can correctly assemble the architecture and produce a model that performs above chance through projection-only training, rather than on matching the published performance.

Several other MLLMs were considered as targets for the reproduction but eventually discarded. Emotion-LLaMA [18] was initially attractive due to its multimodal vision and audio architecture and its domain-specific focus on emotion recognition, which would have aligned well with both Experiment 2 and the system’s modularity claim. It was, however, excluded for two reasons: it requires LoRA fine-tuning alongside the projection layer, which the system does not currently support, and its training dataset is not available on Hugging Face. MiniGPT-4 [86] follows a similar encoder-projection-LLM architecture, but is less extensively benchmarked than LLaVA and uses a more complex two-stage alignment and curation pipeline. mPLUG-Owl [77] has a modular design that

aligns well with the philosophy of the system, but its architecture relies on a separate visual abstractor module and a multi-stage training procedure that adds complexity beyond what is needed to validate the system.

5.4.4 Procedure

In the first step, the LLaVA architecture is declared in the intent configuration by specifying `openai/clip-vit-large-patch14-336` as the vision encoder, a two-layer MLP projection with GELU activations as the projection layer, and `lmsys/vicuna-7b-v1.5` as the LLM.

In the second step, the feature alignment stage is executed using the `liuhaotian/LLaVA-Pretrain` dataset, corresponding to Stage 1 of the LLaVA-1.5 training procedure.

In the third step, the resulting model is evaluated on the same benchmarks reported in the LLaVA-1.5 paper, namely the image subset of ScienceQA and MMBench, under both scoring protocols of Section 5.1.2.

The experiment is considered successful if the full LLaVA-1.5 Stage-1 architecture can be expressed through the declarative intent configuration without custom model code (H5), and if the resulting model performs above random chance on ScienceQA and MMBench under the likelihood-based protocol (H6). A model that meets H6 demonstrates that the projection layer successfully aligns the visual encoder’s output with the LLM’s embedding space, which is the core operation the system automates.

5.4.5 Benchmarks

ScienceQA and MMBench were selected because both are reported in the original LLaVA-1.5 paper [45], which enables a direct comparison between the reproduced model and the published results. Using the same benchmarks as the target model avoids the methodological ambiguity that would arise from evaluating on a different set of tasks. MMBench additionally overlaps with Experiment 1, allowing the evaluation pipeline to be reused and providing a cross-experiment reference point.

ScienceQA [47], in its image subset, is a multiple-choice VQA benchmark that requires multimodal reasoning over scientific diagrams and natural images. The evaluation is

restricted to the questions of the `test` split that include an image, matching the image subset reported in the original paper. The answer options of each question are presented as lettered choices, and the first option letter parsed from the model’s completion is scored against the ground truth, with unparseable outputs counted as incorrect. As for MMBench, a single aggregate accuracy is reported. LLaVA-1.5 (7B) reports 66.8 % accuracy on ScienceQA (image subset) after the full two-stage pipeline [45], so this reproduction’s score is expected to be notably lower.

MMBench [46] provides a comprehensive multi-ability evaluation covering 20 fine-grained dimensions (see Experiment 1). LLaVA-1.5 (7B) reports 64.3 % on MMBench-EN after the full two-stage pipeline [45]. The same single-pass evaluation protocol as in Experiment 1 is applied.

5.4.6 Results

The full LLaVA-1.5 Stage-1 architecture was declared in a single intent configuration, from which the Engine assembled and trained the model once per seed, and no changes to the Engine code base were required at any point. In particular, the exchange of the language backbone from `Qwen/Qwen3-1.7B` to `lmsys/vicuna-7b-v1.5` was expressed by editing one node of the configuration, in the same way as the encoder exchange in Experiment 1. This confirms H5 as the intent configuration can express an independently designed and published MLLM architecture without any custom model code.

As in Experiment 1, the generation-based protocol confirmed the format-compliance failure anticipated in Section 5.1.2, with accuracies below 0.01 on both benchmarks across all seeds. The assessment of H6 therefore rests on the likelihood-based scores. The likelihood-based scores are reported in Table 5.6 as the mean over the three seeds, together with the measured chance baselines and the published LLaVA-1.5 reference scores.

Model	ScienceQA (image) acc.	MMBench acc.
LLaVA-1.5 Stage-1 reproduction	0.414	0.506
Measured chance baseline	0.355	0.266

LLaVA-1.5 (7B), published [45]	0.668	0.643
--------------------------------	-------	-------

Table 5.6: Likelihood-based benchmark accuracies for Experiment 3, averaged over seeds 42, 2, and 534. The chance baselines are measured as the mean number of answer options over the evaluated questions (see Section 5.1.2). The published LLaVA-1.5 (7B) scores are included for orientation only, as they were obtained after the full two-stage training pipeline and under a generation-based protocol.

The reproduced model scores above chance on both benchmarks under every seed. On MMBench, the per-seed accuracies range from 0.493 to 0.513 and thereby reach nearly twice the measured chance level of 0.266. On ScienceQA, the margin is smaller but present under every seed. The per-seed accuracies range from 0.408 to 0.419 against a chance level of 0.355, which is elevated because many questions of the image subset offer only two or three answer options. The per-seed spread is narrow on both benchmarks, 0.012 on ScienceQA and 0.020 on MMBench, and in contrast to Experiment 1 the seed ranking is consistent across benchmarks, with seed 534 producing the weakest score on both.

Both scores fall short of the published values of 0.668 and 0.643, though the two results are not directly comparable. The published scores were obtained under a generation-based protocol (see Section 5.1.2), and the models behind them underwent instruction tuning omitted here (see Section 5.4.4). The published values therefore serve as an orientation rather than as a quantitative baseline, and the quantitative criterion of this experiment remains the comparison against the measured chance level.

To isolate the visual contribution of the learned projection, the same three control conditions described in Section 5.2.5 were applied to the reproduced model. Table 5.7 reports the mean over seeds 2, 42, and 534 for each condition on both benchmarks.

Condition	ScienceQA (image) acc.	MMBench acc.
Trained	0.414	0.506
Shuffled	0.365	0.317
Text-only	0.475	0.297
Random projection	0.347	0.259
Chance	0.355	0.266

Condition	ScienceQA (image) acc.	MMBench acc.
-----------	------------------------	--------------

Table 5.7: Mean likelihood-based accuracy over seeds 2, 42, and 534 under four scoring conditions. The text-only result is identical across all three seeds because suppressing the visual prefix collapses each to the bare Vicuna-7B LLM.

The two benchmarks yield opposite verdicts on whether the model outperforms a blind LLM, and the controls reveal why.

On MMBench, the trained model exceeds the text-only baseline by 20.9 percentage points and the shuffled condition by 18.9 percentage points, while the random-projection condition sits near chance (0.259 against 0.266). The language prior accounts for only 3.1 percentage points above chance (text-only 0.297 against chance 0.266), so the trained margin is attributable to the learned visual alignment. On MMBench, H6 is confirmed.

On ScienceQA, the text-only baseline scores 0.475, which is 12.0 percentage points above chance and 6.1 percentage points above the trained model. Prepending visual tokens through the caption-trained projection costs accuracy on a benchmark whose questions are largely answerable from the text alone. The trained model nevertheless beats the shuffled condition by 4.9 percentage points and the random-projection condition sits at chance (0.347 against 0.355), so the projection does transmit image-specific information. On ScienceQA, the alignment is functional in the sense that the correct image outperforms a wrong one, but the net effect is negative because the text-only baseline already exceeds chance by 12.0 percentage points (0.475 against 0.355), and prepending visual tokens through the projection displaces more of this text-based signal than it adds in visual information.

H6 therefore splits into two claims. The alignment claim, that projection-only training produces a feature alignment that transmits image-specific information, holds on both benchmarks and under every seed. The usefulness claim, that the resulting model outperforms a blind LLM, holds on MMBench and fails on ScienceQA. The difference traces to how much each benchmark can be solved from text alone, 3.1 percentage points above chance on MMBench against 12.0 on ScienceQA, which determines whether the visual information added by the projection exceeds the text-based signal it displaces.

The MMBench score additionally provides the cross-experiment reference point anticipated in the benchmark selection. The reproduction reaches a mean MMBench accuracy of 0.506, whereas the configurations of Experiment 1, evaluated under the identical protocol and chance baseline, reach 0.341 and 0.366 (see Table 5.3). This comparison is directional only, since the two experiments differ in the language backbone (seven billion against 1.7 billion parameters), the alignment corpus (LCS-558K against COCO Captions), and the executing hardware. The direction of the difference is nevertheless consistent with the expectation that a larger language backbone and a larger alignment corpus improve the attainable performance of the feature alignment stage.

6 Discussion

Taken together, the three experiments exercise the system across a non-trivial range of LLMs, encoders, modalities, and datasets. In each case the architectural change was expressed through the JSON intent configuration described in Section 4, with the Engine handling preprocessor selection, projection-dimension matching, and training execution without further manual intervention. Two LLM families at different scales are covered, with `Qwen/Qwen3-1.7B` in Experiments 1 and 2 and `lmsys/vicuna-7b-v1.5` in Experiment 3. The three encoders exercised span two modalities, namely `openai/clip-vit-large-patch14-336` and `google/siglip2-so400m-patch14-384` for vision and `facebook/wav2vec2-base-960h` for audio. The training data varies along the same lines. The vision projection layers of Experiments 1 and 2 are trained on `jxie/coco_captions`, the audio branch of Experiment 2 is aligned on `AudioLLMs/audiocaps_qa_test`, and Experiment 3 uses `liuhaotian/LLaVA-Pretrain` to match the corpus of the reproduced model. Each of these substitutions of the LLM, the encoder, the modality, or the training data required editing only the corresponding node in the intent configuration, while the remainder of the pipeline definition stayed untouched.

Each experiment targets a distinct property of the system. Experiment 1 is a controlled component swap that replaces the CLIP vision encoder with SigLIP-2 while holding the LLM, the dataset, and all user-specified training parameters constant, so that the two resulting configurations differ only in the encoder and the derived projection dimensions. Experiment 2 extends a vision-only baseline with an audio branch, demonstrating that the system can scale from a single-modality to a multimodal architecture through the addition of new nodes in the intent configuration. Experiment 3 substitutes both the LLM and the dataset to reproduce the published LLaVA-1.5 architecture, validating that the system can express real-world MLLM designs and not only the configurations constructed for the preceding experiments.

This breadth of coverage supports the claim of modularity and generalizability of the tool formulated in the research question (see Section 1). The same Engine code base handled all three experiments without modification, demonstrating that the intent-based design separates the “what” of the model architecture from the “how” of its assembly and training.

The development history of the platform provides evidence for this separation. The JSON intent configuration was designed before any Engine code existed and served as the fixed contract between the two components from then on. Export, caching, and numerical precision handling were each revised several times over the course of the work, yet the configuration schema remained largely stable and no revision invalidated an existing pipeline definition. The Designer was consequently developed against the schema alone and remained unaffected by the concurrent rewrites of the Engine internals it never observes.

The interchangeability demonstrated by the experiments also rests on an architectural mechanism that resolves a practical obstacle to component swapping. Encoders and LLMs expose embedding spaces of differing dimensionality, so a naive exchange of either component would break the tensor contract between them. As described in Section 4, the Engine derives the input dimension of each projection layer from the output of the connected encoder and its output dimension from the input embedding size of the connected LLM. Dimension bookkeeping is thereby lifted from the user into the platform, which is what allows the encoder swap of Experiment 1 and the LLM substitution of Experiment 3 to be expressed as edits to a single node while the projection layers adapt automatically.

Dimension matching is, however, only the most visible instance of the bookkeeping the Engine absorbs. The Hugging Face ecosystem exposes no uniform contract between models, preprocessors, and datasets. Preprocessors disagree on the names of their input arguments, audio datasets differ in how their waveforms are decoded, compound encoders do not indicate which of their sub-models determines the relevant embedding dimension, and dataset columns frequently require casts before they can be used. A considerable part of the Engine code therefore consists of adapter logic that resolves such differences case by case, in one instance by invoking a preprocessor experimentally and parsing the resulting error message to discover the expected argument name. Absorbing this heterogeneity constituted one of the largest engineering efforts of the project, and it is exactly this effort that the intent-based abstraction removes from the user.

Looking at the hypotheses formulated in Section 5, the results divide into two groups with different outcomes. The workflow hypotheses H1, H3, and H5, which assert that a component replacement, a modality extension, and the reproduction of a published architecture can each be expressed purely through the intent configuration, are confirmed. The performance hypotheses show a more differentiated picture. H6 is partially confirmed. The control conditions of Table 5.7 establish that the learned projection transmits image-specific information on both benchmarks, as the trained model outperforms both the shuffled condition and the random-projection condition, whose scores sit at chance. On MMBench the trained model additionally exceeds the text-only baseline by over 20 percentage points, confirming that the visual alignment produces a benefit. On ScienceQA the text-only baseline scores above the trained model, because the text-only accuracy already exceeds chance by 12 percentage points and prepending visual tokens displaces more of this text-based signal than it adds (see Table 5.7). H2 is supported, as SigLIP-2 outperforms CLIP on MMBench (see Table 5.3). H4 is not confirmed, because providing the audio input at inference lowers the caption quality under every seed tested instead of preserving it (see Table 5.5).

This split carries the central lesson of the experimental part. Declarative composition succeeded in every case, as each declared architecture was assembled, trained, and delivered as a functional model. Yet the resulting models did not uniformly benefit from the architectural changes expressed, which shows that mechanical composability guarantees that a configuration is realizable, not that it is advantageous. The platform consequently shifts the effort of MLLM construction away from implementation and toward the experimental validation of the composed design, which is precisely the iteration loop the intent-based approach is meant to accelerate.

How such validation must be conducted is itself a finding of the experiments. The generation-based protocol proved uninformative for every configuration under test. Projection layers aligned solely on caption data respond to multiple-choice prompts with descriptive prose, so the near-zero generation-based scores observed in Experiments 1 and 3 measure answer-format compliance rather than visual capability. For models that have not been instruction-tuned, the likelihood-based protocol of Section 5.1.2 is therefore not a methodological convenience but a prerequisite for interpretable scores. The seed sweeps proved equally important. In Experiment 1 the three-seed sweep was essential to establishing H2, as it revealed non-overlapping per-seed ranges on MMBench that a single run could not have demonstrated. The narrow and consistently ranked per-seed spread of Experiment 3 shows that this variability is not a property of the evaluation

pipeline itself but of the small effect sizes at this training scale. Benchmark comparisons between projection-only models should consequently be based on repeated runs, and single-run results treated as indicative at best.

The control conditions reported in Table 5.4 and Table 5.7 add a third methodological lesson. Scoring above the uniform-chance baseline does not, by itself, establish that a model uses the image, because the likelihood-based protocol ranks answer options under the LLM, and the language prior can carry a question without any visual input. This is the critique raised by Chen et al. [15] for MMBench and ScienceQA in the context of their MMStar benchmark. The text-only baselines measured here confirm it in this thesis’s own setup, with the text-only score exceeding chance by 12 percentage points on ScienceQA against only 3 percentage points on MMBench. The shuffled and random-projection conditions then isolate the visual contribution, which is present on both benchmarks but masked on ScienceQA by the larger text-based signal it displaces. Reporting above-chance scores without a blind baseline is therefore insufficient for any claim about visual alignment, and future work with the platform should include such controls as a standard part of the evaluation.

A further observation of the same kind concerns failure modes that surface only in long training runs. Several defects passed every short validation run and appeared after thousands of steps, among them the fp16 overflow described in Section 5.1.1 and evaluation runs that silently produced all-zero scores instead of raising an error. The Engine therefore gained explicit guards over the course of the experiments, including a finiteness check on the training loss, a rejection of degenerate evaluation results, and the logging of raw model outputs alongside the computed scores. These guards were introduced reactively, yet without them the reported results could not have been trusted. Trustworthy comparisons further required the pipeline to be made deterministic, with fixed seeds propagated to every library involved and all Hugging Face models and datasets pinned to specific revisions.

A comparable lesson emerged from the delivery of the trained models. Because the resulting artifact is not a single network but a graph of encoders, projection layers, and an LLM held together by orchestration logic, established export paths turned out not to be compatible. Both `torch.export` and ONNX were tried, but both failed on the dynamic input shapes and the non-serializable structures that multimodal composition produces. The export mechanism described in Section 4, a custom Python runtime combined with a safetensors file of the trainable weights, is thus not a first choice but

the conclusion of an elimination process. The absence of a standard format able to represent compositional MLLMs is taken up again as an open problem in the following chapter.

The two performance shortfalls, finally, share a common origin in the training regime rather than constituting independent defects. Both the audio-induced degradation of Experiment 2 and the distance between the reproduction of Experiment 3 and its published reference trace back to the same design boundary. The Engine aligns each projection layer in isolation against frozen backbones, and no jointly multimodal training stage or instruction tuning follows. In Experiment 2, this sequential per-branch alignment means that each projection layer is only ever optimized in a context where the other modality slot holds its learned missing-modality embedding, so the joint input presented at inference is a configuration through which no gradient ever passed and for which the two projection layer outputs were never calibrated to coexist. In Experiment 3, the omission of the visual instruction tuning stage, which in the original pipeline jointly fine-tunes the projection layer and the LLM, leaves the reproduction functional but necessitates likelihood-based scoring in place of the generation-based protocol used by the original authors, so the two sets of scores are not directly comparable. This boundary is the direct consequence of the frozen-backbone invariant described in Section 5.1.1. The experiments quantify the cost of this trade-off.

The observed performance levels are partly a consequence of the experimental design prioritizing controlled comparison over optimized quality. The training hyperparameters, including learning rate, projection depth, and step budget, were held constant across all configurations to isolate the variable under test rather than tuned for best achievable performance. Within the frozen-backbone paradigm itself, there is room for improvement through hyperparameter optimization, larger or more diverse training corpora, and deeper or wider projection architectures. Beyond projection-only training, instruction tuning via parameter-efficient methods such as LoRA [33] would extend the system without abandoning its core design properties, since automatic dimension matching and the declarative configuration remain intact when only adapter weights are added to the trainable surface. The frozen-backbone constraint therefore represents the floor of what the current prototype demonstrates, not the ceiling of what the intent-based approach can deliver.

Returning to the research questions posed in Section 1.2, the experiments permit a direct answer to each. RQ1 asked whether a modular, intent-based platform can enable the

composition, extension, and component-by-component replacement of MLLMs without direct code changes. The three experiments confirm that it can. Every architectural change tested, an encoder replacement, a modality extension, and the reproduction of a published architecture, was expressed entirely through the JSON intent configuration, and the same Engine code base handled all three without modification. The composition, replacement, and extension of the model itself require no code changes, while ingesting a dataset whose layout the column-mapping mechanism does not cover still requires a per-dataset preparation step outside the platform.

RQ2 asked whether the models produced by the platform yield functionally adequate outputs, in the sense of scoring measurably above chance on multimodal evaluation tasks. The answer is partially affirmative and qualified by the control conditions. The experiments establish that projection-only training produces a feature alignment that transmits image-specific information in every configuration tested, as trained models consistently outperform both the shuffled and random-projection conditions (see Table 5.4 and Table 5.7). Whether this alignment translates into a benefit over a blind LLM depends on the benchmark. On MMBench, where the language prior accounts for only a small margin above chance, both Experiment 1 and Experiment 3 show clear gains from the visual pathway. On ScienceQA, where the language prior alone reaches 12 percentage points above chance, the visual prefix displaces more text-based signal than it adds, and the trained model scores below the text-only baseline. The encoder swap of Experiment 1 produces a consistent improvement on MMBench with non-overlapping per-seed ranges, supporting H2. The multimodal extension of Experiment 2 did not preserve the vision pathway’s performance when the audio input was provided at inference, so H4 is not confirmed. Taken together, projection-only alignment reliably produces a functional feature alignment between frozen encoders and frozen LLMs, but whether that alignment improves over a blind baseline depends on how much the benchmark can already be solved from text alone, and the calibration of multimodal interplay lies beyond what the current single-branch training mode can deliver.

7 Limitations and Outlook

The current system provides a functional prototype that demonstrates the viability of intent-based MLLM construction, but the prototype is bounded by a number of architectural and practical constraints. This chapter first describes these limitations and then outlines directions for future work.

7.1 Limitations

The system imposes several architectural constraints that bound the space of models it can express. The LLM component must be a causal language model with a decoder-only architecture, such as LLaMA, Qwen, or GPT, that exposes `get_input_embeddings()` and `lm_head`. Sequence-to-sequence architectures such as T5 or BART and encoder-only models such as BERT are not supported, as the projection-injection mechanism is designed specifically for the causal decoder forward pass. The projection layers follow a fixed MLP design with GELU activations. While the depth and width of this MLP can be configured through the configuration, alternative architectures such as Q-Former-style cross-attention [42], residual connections, or different normalization strategies cannot be expressed. Furthermore, the system relies exclusively on the Hugging Face Transformers ecosystem for model and dataset access. Only models that can be downloaded and executed through the Hugging Face API are supported, and access to gated models requires a valid user access token from Hugging Face. Because the Engine loads encoder weights using Hugging Face’s `trust_remote_code` mechanism, users must be aware that this executes arbitrary Python code provided by the model author, which introduces a potential security consideration.

The reliance on Hugging Face also extends to the structure of the training data. Datasets on Hugging Face do not follow a common schema, as each dataset defines its own, sometimes unique, layout of columns and nested structures. The column-mapping mechanism

of the dataset node covers datasets whose samples can be expressed as flat input and target columns, but it cannot ingest arbitrary layouts. For the experiments in this work, the training corpora had to be reshaped into this flat format in a preparation step before training. The system can therefore not consume every dataset published on Hugging Face out of the box, and a general solution for arbitrary dataset formats remains out of scope for the current prototype. The reliability of the public dataset infrastructure proved to be a constraint of its own. Streamed downloads failed intermittently during the experiments, so the streaming-first design had to be supplemented with retry logic and the local materialization cache described in Section 5.1.1 before long training runs became dependable.

The training pipeline carries additional constraints. Only projection layers are trained while all backbone components remain frozen. When a configuration declares several modalities, the Engine aligns each projection layer sequentially on its own dataset and fills the remaining modality slots with the learned missing-modality embedding (see Section 4.3). As analyzed in Section 6, this training regime cannot match the performance of approaches that jointly fine-tune the LLM alongside the projection layers and leaves the joint use of several modalities at inference uncalibrated. Training datasets in streaming mode are iterated sequentially without shuffling, which can lead to correlated gradient updates for datasets with inherent ordering, such as those sorted by label or timestamp. The only metric computed during training is cross-entropy loss. Task-specific evaluation metrics such as BLEU, VQA accuracy [27], or word error rate are not integrated, which limits the ability to assess model quality during the training process itself. The learning rate schedule uses linear warmup followed by cosine decay, with the total step count derived from the actual dataset length or, where a step limit is configured, from that limit. Only when a dataset is streamed without a determinable length and no step limit is set does the scheduler fall back to a constant learning rate after warmup, which lacks the stabilizing effect of decay over long runs. No maximum token length is enforced through truncation of text inputs before collation, meaning a single unusually long sample can cause out-of-memory errors for the entire batch. The training loop also lacks checkpointing, so an interrupted run cannot be resumed and must be restarted from the beginning.

Reproducibility is additionally bounded by the reliance on a live model hub. Results are repeatable only when the models and datasets involved are pinned to specific revisions, as unpinned references may resolve to different contents over time. Support for this

pinning was added to the intent configuration and is therefore available but not enforced. Configurations without pinned revisions remain non-reproducible.

Hardware requirements also represent a practical constraint. Experiments 1 and 2 were conducted on a single NVIDIA A100 with 40 GB of VRAM, which is the minimum configuration required to run the system with the models tested, while Experiment 3 was executed on a single NVIDIA H100 NVL with 94 GB of VRAM due to the availability of compute resources at the time. Users with less capable hardware may be unable to train configurations that include larger LLMs or high-resolution vision encoders.

Finally, the experiments support the claim of generalizability of the tool but do not cover every case. Only one audio encoder is exercised, and both vision encoders belong to the CLIP/SigLIP family. LLM scaling is not tested beyond 7B, and the trained projection layers are not evaluated for cross-dataset transfer. These are scope decisions consistent with a tooling contribution rather than a model-quality study, but they bound the strength of any generalization argument that could be drawn from the results.

7.2 Outlook

Several of these limitations translate directly into directions for future work.

The training pipeline would benefit most from extensions that bring it closer to the multi-stage procedures used by published MLLMs. Introducing multi-stage training, where projection layers are first pre-trained on large-scale alignment data and then fine-tuned on task-specific corpora, would enable higher-quality models, as demonstrated by the two-stage procedures of LLaVA [44] and mPLUG-Owl [77]. The same applies to joint fine-tuning of the projection layers together with the LLM via parameter-efficient methods such as LoRA [33], which mPLUG-Owl employs in its second training stage. Adding a dedicated instruction tuning stage, whether through LoRA or through fine-tuning of the projection layers on instruction-following data, would address both the answer-format compliance failures observed in Experiments 1 and 3 and the gap between the Experiment 3 reproduction and its published reference scores. Systematic hyperparameter optimization, such as learning rate sweeps, projection depth and width searches, and optimizer selection, would further improve model quality within the current frozen-backbone paradigm without requiring architectural changes to the Engine. Related improvements include early stopping based on validation loss and alternative

loss functions. Data augmentation, such as random cropping and color jitter applied before the vision preprocessor, would expose the projection layers to a wider region of the embedding space produced by the frozen encoders and could improve the robustness of the learned alignment. When multiple datasets are configured, they are currently trained sequentially rather than interleaved, which risks catastrophic forgetting of earlier modality alignments. An interleaved or mixed-batch sampling strategy would mitigate this risk. Adding periodic checkpoint saves would address the missing resumability and make long training jobs more resilient. Gradient checkpointing on the projection layers themselves would reduce GPU memory consumption and enable training of deeper projection architectures on memory-constrained hardware. Finally, explicit multi-GPU support through data-parallel training would allow the system to scale beyond the single-GPU configurations tested in this work.

On the frontend side, several features would lower the barrier to entry for non-expert users. A dataset inspection feature that detects the contained modalities and suggests suitable column mappings when a dataset node is configured would reduce configuration errors. Similarly, integrating search and autocomplete for Hugging Face model IDs and dataset column names would remove the requirement for users to know exact identifiers. A template library with importable model configurations for common architectures (e.g., CLIP vision encoder + Whisper audio encoder + Qwen LLM) would accelerate experimentation. A pre-export validation pass that detects orphaned nodes, missing connections, and incomplete configurations would prevent failed training runs that currently only produce console warnings. Support for uploading local dataset files in formats such as CSV, JSON, or Parquet would allow users to train on private datasets without first publishing them to Hugging Face. Usability would further benefit from session persistence with auto-save and an undo/redo history stack, automated build-status polling with progress indicators, and hyperparameter presets with contextual help tooltips.

The training monitoring capabilities could also be extended. Validation loss is currently only reported as a numeric value in the status stream. Integrating a live loss curve, per-modality accuracy metrics, or support for standard benchmarks would give users actionable feedback on whether the projection layers are converging and would narrow the evaluation gap described above. The guards that were added reactively during the experiments (see Section 6) point in the same direction. A platform that trains user-defined architectures should provide such guardrails by default, because its users will encounter the same silent failure modes without the diagnostic context that was available during this work.

From an architectural perspective, the most impactful extension would be a plugin system for new node types and modalities. The Engine already accepts arbitrary modality strings and attempts to resolve a suitable Hugging Face preprocessor generically, and the Designer likewise treats the modality as a free-form label. What is missing is a mechanism for third parties to contribute encoder types or preprocessing logic that the Hugging Face ecosystem does not provide, without modifying the core code base. A plugin registry of this kind would directly serve the extensibility aspect of the research question formulated in Section 1.2, which asks for a platform that enables the extension of MLLMs without direct code changes. The Engine currently also assumes exactly one LLM node per pipeline. Supporting multiple LLMs for ensemble outputs or routing architectures, as well as recurrent connections, would unlock more expressive model topologies.

Finally, the export and deployment pipeline presents further opportunities. The exported runtime currently processes one sample at a time with no encoder output caching. Batched inference, caching of encoder embeddings across multiple text queries for the same multimodal input, and streaming token generation would improve the practicality of deployed models. Because the trained artifact is a multi-component graph with custom orchestration logic, no existing standard format (GGUF, LoRA adapters, ONNX) can represent it directly. Developing a standardized, versioned export format or contributing the architecture as a first-class Hugging Face model type would improve reproducibility and simplify sharing across teams. Neither the export format nor the JSON pipeline configuration currently carries a version identifier, meaning older configurations may silently break as the schema evolves. Adding a `schema_version` field would enable forward-compatible parsing and migration paths. Enabling revision pinning by default would complement such a version field, turning a pipeline configuration into a reproducible artifact rather than a reference into a moving ecosystem.

Bibliography

- [1] PyTorch. <https://pytorch.org/>. (accessed 2026-07-28).
- [2] React. <https://react.dev/>. (accessed 2025-11-20).
- [3] Welcome to Pydantic - Pydantic Validation. <https://docs.pydantic.dev/latest/>. (accessed 2025-11-28).
- [4] React FLOW. <https://reactflow.dev/learn>, October 2025. (accessed 2025-11-07).
- [5] Unslothai/unsloth. Unsloth AI, July 2026.
- [6] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob L Menick, Sebastian Borgeaud, Andy Brock, Aida Nematzadeh, Sahand Sharifzadeh, Miłkoł aj Bińkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karén Simonyan. Flamingo: A Visual Language Model for Few-Shot Learning. In *Advances in Neural Information Processing Systems*, volume 35, pages 23716–23736. Curran Associates, Inc., 2022.
- [7] Hadi Amini, Md Jueal Mia, Yasaman Saadati, Ahmed Imteaj, Seyed sina Nabavi-razavi, Urmish Thakker, Md Zarif Hossain, Awal Ahmed Fime, and S. S. Iyengar. Distributed LLMs and Multimodal Large Language Models: A Survey on Advances, Challenges, and Future Directions, March 2025.
- [8] Jisu An, Junseok Lee, Jeoungeun Lee, and Yongseok Son. Towards LLM-Centric Multimodal Fusion: A Survey on Integration Strategies and Techniques, June 2025.
- [9] Vaastav Anand, Yichen Li, Alok Gautam Kumbhare, Celine Irvine, Chetan Bansal, Gagan Somashekar, Jonathan Mace, Pedro Las-Casas, Ricardo Bianchini, and Rodrigo Fonseca. Intent-based System Design and Operation. In *Proceedings of the*

- 4th Workshop on Practical Adoption Challenges of ML for Systems*, pages 81–86, Seoul Republic of Korea, October 2025. ACM. ISBN 979-8-4007-2205-9. doi: 10.1145/3766882.3767182.
- [10] William Beauchamp, Colin Simpson, Roderick Bramblewick, Andrew Scolto, and Lucas Brackenridge. Emergent Modal Fusion Through Multilevel Subsymbolic Routing in Large Language Models. 2025.
- [11] Michael H. Behringer, Max Pritikin, Steinthor Bjarnason, Alexander Clemm, Brian E. Carpenter, Sheng Jiang, and Laurent Ciavaglia. Autonomic Networking: Definitions and Design Goals. Request for Comments RFC 7575, Internet Engineering Task Force, June 2015.
- [12] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A Neural Probabilistic Language Model. 2003.
- [13] Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Jeffrey Hsu, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, Niklas Muennighoff, Ellie Pavlick, Jason Phang, Aviya Skowron, Samson Tan, Xiangru Tang, Kevin A. Wang, Genta Indra Winata, François Yvon, and Andy Zou. Lessons from the Trenches on Reproducible Evaluation of Language Models, May 2026.
- [14] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, July 2020.
- [15] Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, and Feng Zhao. Are We on the Right Way for Evaluating Large Vision-Language Models? In *Advances in Neural Information Processing Systems*, volume 37, pages 27056–27087. Curran Associates, Inc., 2024. doi: 10.52202/079017-0850.

- [16] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollar, and C. Lawrence Zitnick. Microsoft COCO Captions: Data Collection and Evaluation Server, April 2015.
- [17] Yunkai Chen, Qimeng Wang, Shiwei Wu, Yan Gao, Tong Xu, and Yao Hu. TOMGPT: Reliable Text-Only Training Approach for Cost-Effective Multi-modal Large Language Model. *ACM Transactions on Knowledge Discovery from Data*, 18(7):1–19, August 2024. ISSN 1556-4681, 1556-472X. doi: 10.1145/3654674.
- [18] Zebang Cheng, Zhi-Qi Cheng, Jun-Yan He, Kai Wang, Yuxiang Lin, Zheng Lian, Xiaojiang Peng, and Alexander Hauptmann. Emotion-LLaMA: Multimodal Emotion Recognition and Reasoning with Instruction Tuning. *Advances in Neural Information Processing Systems*, 37:110805–110853, December 2024.
- [19] Alexander Clemm, Laurent Ciavaglia, Lisano Zambenedetti Granville, and Jeff Tantsura. Intent-Based Networking - Concepts and Definitions. Request for Comments RFC 9315, Internet Engineering Task Force, October 2022.
- [20] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language Modeling with Gated Convolutional Networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 933–941. PMLR, July 2017.
- [21] Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. Fine-Tuning Pretrained Language Models: Weight Initializations, Data Orders, and Early Stopping, February 2020.
- [22] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, June 2021.
- [23] Michael R. Douglas. Large Language Models, October 2023.
- [24] Jeffrey L. Elman. Finding Structure in Time. *Cognitive Science*, 14(2):179–211, 1990. ISSN 1551-6709. doi: 10.1207/s15516709cog1402_1.
- [25] Christian Endres, Uwe Breitenbucher, Michael Falkenthal, Oliver Kopp, Frank Leymann, and Johannes Wettinger. Declarative vs. Imperative: Two Modeling Patterns for the Automated Deployment of Applications. 2017.

- [26] Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra. ImageBind: One Embedding Space To Bind Them All, May 2023.
- [27] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6904–6913, 2017.
- [28] Pankaj Kumar Gupta. Intent-Based Networking Architecture: A Deep Dive into Its Components and Workflow. 2025.
- [29] Xiaotian Han, Yiqi Wang, Bohan Zhai, Quanzeng You, and Hongxia Yang. COCO is "ALL" You Need for Visual Instruction Fine-tuning, January 2024.
- [30] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring Massive Multitask Language Understanding, January 2021.
- [31] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.
- [32] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-Efficient Transfer Learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2790–2799. PMLR, May 2019.
- [33] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models, October 2021.
- [34] Dawei Huang, Chuan Yan, Qing Li, and Xiaojiang Peng. From Large Language Models to Large Multimodal Models: A Literature Review. *Applied Sciences*, 14(12):5068, January 2024. ISSN 2076-3417. doi: 10.3390/app14125068.
- [35] Kai Huang, Boyuan Yang, and Wei Gao. Modality Plug-and-Play: Elastic Modality Adaptation in Multimodal LLMs for Embodied AI, December 2023.
- [36] Chris Dongjoo Kim, Byeongchang Kim, Hyunmin Lee, and Gunhee Kim. AudioCaps: Generating Captions for Audios in The Wild. In Jill Burstein, Christy Doran, and Tamar Solorio, editors, *Proceedings of the 2019 Conference of the*

- North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 119–132, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1011.
- [37] Jing Yu Koh, Ruslan Salakhutdinov, and Daniel Fried. Grounding Language Models to Images for Multimodal Inputs and Outputs. In *Proceedings of the 40th International Conference on Machine Learning*, pages 17283–17300. PMLR, July 2023.
 - [38] Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing, August 2018.
 - [39] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. ISSN 1558-2256. doi: 10.1109/5.726791.
 - [40] Aris Leivadeas and Matthias Falkner. A Survey on Intent-Based Networking. *IEEE Communications Surveys & Tutorials*, 25(1):625–655, 2023. ISSN 1553-877X. doi: 10.1109/COMST.2022.3215919.
 - [41] Bohao Li, Rui Wang, Guangzhi Wang, Yuying Ge, Yixiao Ge, and Ying Shan. SEED-Bench: Benchmarking Multimodal LLMs with Generative Comprehension. <https://arxiv.org/abs/2307.16125v2>, July 2023. (accessed 2026-07-24).
 - [42] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models. In *Proceedings of the 40th International Conference on Machine Learning*, pages 19730–19742. PMLR, July 2023.
 - [43] Zijing Liang, Yanjie Xu, Yifan Hong, Penghui Shang, Qi Wang, Qiang Fu, and Ke Liu. A Survey of Multimodal Large Language Models. In *Proceedings of the 3rd International Conference on Computer, Artificial Intelligence and Control Engineering*, CAICE '24, pages 405–409, New York, NY, USA, August 2024. Association for Computing Machinery. ISBN 979-8-4007-1694-2. doi: 10.1145/3672758.3672824.
 - [44] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 34892–34916. Curran Associates, Inc., 2023.

- [45] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved Baselines with Visual Instruction Tuning, May 2024.
- [46] Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, Kai Chen, and Dahua Lin. MM-Bench: Is Your Multi-modal Model an All-around Player?, August 2024.
- [47] Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to Explain: Multimodal Reasoning via Thought Chains for Science Question Answering. *Advances in Neural Information Processing Systems*, 35:2507–2521, December 2022.
- [48] Gen Luo, Yiyi Zhou, Tianhe Ren, Shengxin Chen, Xiaoshuai Sun, and Rongrong Ji. Cheap and Quick: Efficient Vision-Language Instruction Tuning for Large Language Models, October 2023.
- [49] Jie Ma, Pinghui Wang, Dechen Kong, Zewei Wang, Jun Liu, Hongbin Pei, and Junzhou Zhao. Robust Visual Question Answering: Datasets, Methods, and Future Challenges. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(8):5575–5594, August 2024. ISSN 1939-3539. doi: 10.1109/TPAMI.2024.3366154.
- [50] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space, September 2013.
- [51] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.
- [52] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large Language Models: A Survey, March 2025.
- [53] OpenAI. GPT-4 Technical Report, March 2023.
- [54] Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling Neural Machine Translation. In Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Christof Monz, Matteo Negri, Aurélie Névoul, Mariana Neves, Matt Post, Lucia Specia, Marco Turchi, and Karin Verspoor, editors, *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 1–9,

- Brussels, Belgium, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-6301.
- [55] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, page 311, Philadelphia, Pennsylvania, 2001. Association for Computational Linguistics. doi: 10.3115/1073083.1073135.
 - [56] Kyle Penfound, Dagger, Cortney Nickerson, and Kubeshop. Production-Grade Container Orchestration. <https://kubernetes.io/>. (accessed 2025-11-19).
 - [57] Gorjan Radevski, Teodora Popordanoska, Matthew B. Blaschko, and Tinne Tuytelaars. DAVE: Diagnostic benchmark for Audio Visual Evaluation, December 2025.
 - [58] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for Activation Functions, October 2017.
 - [59] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Erscheinungsort nicht ermittelbar, 3. aufl edition, 2016. ISBN 978-1-292-15396-4.
 - [60] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162.
 - [61] Noam Shazeer. GLU Variants Improve Transformer, February 2020.
 - [62] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face, December 2023.
 - [63] Shezheng Song, Xiaopeng Li, Shasha Li, Shan Zhao, Jie Yu, Jun Ma, Xiaoguang Mao, and Weimin Zhang. How to Bridge the Gap between Modalities: Survey on Multimodal Large Language Model, January 2025.
 - [64] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and Efficient Foundation Language Models, February 2023.

- [65] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Rannan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open Foundation and Fine-Tuned Chat Models, July 2023.
- [66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
- [67] Shakti N. Wadekar, Abhishek Chaurasia, Aman Chadha, and Eugenio Culurciello. The Evolution of Multimodal Model Architectures, May 2024.
- [68] Han Wang, Yongjie Ye, Bingru Li, Yuxiang Nie, Jinghui Lu, Jingqun Tang, Yanjie Wang, and Can Huang. Vision as LoRA, March 2025.
- [69] Jiaqi Wang, Hanqi Jiang, Yiheng Liu, Chong Ma, Xu Zhang, Yi Pan, Mengyuan Liu, Peiran Gu, Sichen Xia, Wenjun Li, Yutong Zhang, Zihao Wu, Zhengliang Liu, Tianyang Zhong, Bao Ge, Tuo Zhang, Ning Qiang, Xintao Hu, Xi Jiang, Xin Zhang, Wei Zhang, Dinggang Shen, Tianming Liu, and Shu Zhang. A Comprehensive Review of Multimodal Large Language Models: Performance and Challenges Across Different Tasks, August 2024.
- [70] Weiyao Wang, Du Tran, and Matt Feiszli. What Makes Training Multi-Modal Classification Networks Hard?, April 2020.
- [71] Xinpeng Wang, Bolei Ma, Chengzhi Hu, Leon Weber-Genzel, Paul Röttger, Frauke Kreuter, Dirk Hovy, and Barbara Plank. “My Answer is C”: First-Token Prob-

- abilities Do Not Match Text Answers in Instruction-Tuned Language Models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 7407–7416, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.441.
- [72] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-Art Natural Language Processing. Association for Computational Linguistics, October 2020.
- [73] Tianyu Xie, Yuhang Wu, Yongdong Luo, Jiayi Ji, and Xiawu Zheng. Training-Free Multimodal Large Language Model Orchestration, August 2025.
- [74] Dejing Xu, Zhou Zhao, Jun Xiao, Fei Wu, Hanwang Zhang, Xiangnan He, and Yueting Zhuang. Video Question Answering via Gradually Refined Attention over Appearance and Motion. In *Proceedings of the 25th ACM International Conference on Multimedia*, pages 1645–1653, Mountain View California USA, October 2017. ACM. ISBN 978-1-4503-4906-2. doi: 10.1145/3123266.3123427.
- [75] Jun Xu, Tao Mei, Ting Yao, and Yong Rui. MSR-VTT: A Large Video Description Dataset for Bridging Video and Language. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5288–5296, Las Vegas, NV, USA, June 2016. IEEE. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.571.
- [76] Qinghao Ye, Haiyang Xu, Jiabo Ye, Ming Yan, Anwen Hu, Haowei Liu, Qi Qian, Ji Zhang, Fei Huang, and Jingren Zhou. mPLUG-Owl2: Revolutionizing Multimodal Large Language Model with Modality Collaboration, November 2023.
- [77] Qinghao Ye, Haiyang Xu, Guohai Xu, Jiabo Ye, Ming Yan, Yiyang Zhou, Junyang Wang, Anwen Hu, Pengcheng Shi, Yaya Shi, Chenliang Li, Yuanhong Xu, Hehong Chen, Junfeng Tian, Qi Qian, Ji Zhang, Fei Huang, and Jingren Zhou. mPLUG-Owl: Modularization Empowers Large Language Models with Multimodality, March 2024.
- [78] Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. A survey on multimodal large language models. *National Science Review*, 11(12):nwae403, December 2024. ISSN 2095-5138. doi: 10.1093/nsr/nwae403.

- [79] Henry Yu, Hesam Rahimi, Christopher Janz, Dong Wang, Zhen Li, Chungang Yang, and Yehua Zhao. Building a Comprehensive Intent-Based Networking Framework: A Practical Approach from Design Concepts to Implementation. *Journal of Network and Systems Management*, 32(3):47, May 2024. ISSN 1573-7705. doi: 10.1007/s10922-024-09819-7.
- [80] Engin Zeydan and Yekta Turk. Recent Advances in Intent-Based Networking: A Survey. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–5, May 2020. doi: 10.1109/VTC2020-Spring48590.2020.9128422.
- [81] Duzhen Zhang, Yahan Yu, Jiahua Dong, Chenxing Li, Dan Su, Chenhui Chu, and Dong Yu. MM-LLMs: Recent Advances in MultiModal Large Language Models, May 2024.
- [82] Fangyuan Zhang, Tingting Liang, Zhengyuan Wu, and Yuyu Yin. PILL: Plug Into LLM with Adapter Expert and Attention Gate, November 2023.
- [83] Renrui Zhang, Jiaming Han, Chris Liu, Peng Gao, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, and Yu Qiao. LLaMA-Adapter: Efficient Fine-tuning of Language Models with Zero-init Attention, September 2024.
- [84] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. OPT: Open Pre-trained Transformer Language Models, June 2022.
- [85] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A Survey of Large Language Models, March 2025.
- [86] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. MiniGPT-4: Enhancing Vision-Language Understanding with Advanced Large Language Models, October 2023.

A Appendix

A.1 Tools Used

Table A.1 lists the tools and aids used while working on the topic of this Master thesis.

Tool	Usage
L ^A T _E X	Typesetting and layout tool used to create this document
Gemini	Writing style, grammar, translation and research
Claude	Writing style, grammar, translation and research
Zotero	Reference management
Google Scholar	Literature search and research
NotebookLM	Literature search and research

Table A.1: Tools and Aids Used

A.2 Full JSON Configuration Example

Listing A.1 shows the complete JSON Intent exported from the two-modality design in Figure 4.3, of which Listing 4.1 presents the vision path. A vision and an audio modality path, each consisting of a dataset, an encoder, and a projection node, converge on a shared LLM backbone. The LLM node additionally pins a **revision**, fixing the referenced Hugging Face artifact to a specific commit.

```
1 {
2   "project_meta": {
3     "id": "thesis-demo-project-001",
4     "name": "MLLM_Project",
5     "created_at": "2025-11-27T14:00:00Z"
6   },
7   "training_config": {
8     "trust_remote_code": false,
9     "export_formats": ["runtime"],
10    "max_seq_length": 2048,
11    "epochs": 2,
12    "batch_size": 16,
```

```

13     "learning_rate": null,
14     "accumulation_steps": 2,
15     "max_steps": null,
16     "validation_ratio": null,
17     "eval_dataset_id": null,
18     "seed": 42,
19     "use_subprocess": false
20 },
21 "nodes": [
22     {
23         "id": "dataset_vision_1",
24         "type": "dataset",
25         "dataset_id": "jxie/coco_captions",
26         "column_mapping": {
27             "input_columns": { "image": "vision" },
28             "target_column": "caption"
29         },
30         "revision": null
31     },
32     {
33         "id": "dataset_audio_1",
34         "type": "dataset",
35         "dataset_id": "AudioLLMs/audiocaps_qa_test",
36         "column_mapping": {
37             "input_columns": { "context": "audio", "instruction": "text" },
38             "target_column": "answer"
39         },
40         "revision": null
41     },
42     {
43         "id": "encoder_vision_1",
44         "type": "encoder",
45         "model_id": "google/siglip2-so400m-patch14-384",
46         "modality": "vision",
47         "input_arg": "pixel_values",
48         "revision": null
49     },
50     {
51         "id": "encoder_audio_1",
52         "type": "encoder",
53         "model_id": "facebook/wav2vec2-base-960h",
54         "modality": "audio",
55         "input_arg": "input_values",
56         "revision": null
57     },
58     {
59         "id": "bridge_vision_1",
60         "type": "projection",
61         "architecture": { "layers": 2, "nodes_per_layer": 4096 },
62         "inputs": {
63             "source_model": "encoder_vision_1",
64             "training_data": "dataset_vision_1"
65         }
66     },
67     {
68         "id": "bridge_audio_1",
69         "type": "projection",
70         "architecture": { "layers": 1, "nodes_per_layer": 4096 },
71         "inputs": {
72             "source_model": "encoder_audio_1",
73             "training_data": "dataset_audio_1"

```

```
74     }  
75   },  
76   {  
77     "id": "llm_1",  
78     "type": "llm",  
79     "model_id": "Qwen/Qwen3-1.7B",  
80     "load_in_4bit": false,  
81     "max_seq_length": 1024,  
82     "inputs": { "connected_bridges": ["bridge_vision_1", "bridge_audio_1"] },  
83     "revision": "70d244cc86ccca08cf5af4e1e306ecf908b1ad5e"  
84   }  
85 ]  
86 }
```

Listing A.1: Complete JSON configuration exported from the two-modality design shown in Figure 4.3.

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort

Datum

Unterschrift im Original